Univerza v Ljubljani
Fakulteta za računalništvo in informatiko

Jurij Slabanja

Stabilizacija kvadrokopterja z vizualnim sledenjem oznaki

diplomsko delo

univerzitetni študijski program prve stopnje
računalništvo in informatika

Mentor: doc. dr. Danijel Skočaj

Ljubljana 2014

Besedilo je oblikovano z urejevalnikom besedil LaTeX.

Fakulteta za računalništvo in informatiko izdaja naslednjo nalogo:

Tematika naloge:

V diplomski nalogi implementirajte algoritem, ki bo omogočal boljšo stabilizacijo kvadrokopterja AR.Drone. Kvadrokopter je opremljen z dvema kamerama. S procesiranjem zajetih slik naj razviti algoritem zaznava oznako na tleh, na osnovi zaznane oznake pa naj ustrezno krmili kvadrokopter, da bo lebdel nad oznako. Oznako namestite tudi na mobilno platformo TurtleBot, ki se giblje po prostoru, kvadrokopter pa naj mobilni platformi sledi. Razviti sistem ustrezno ovrednotite; ocenite kako uspešno lahko sledi oznaki ter primerjajte stabilnost lebdenja s stabilnostjo, ki jo zagotavlja privzeto delovanje kvadrokopterja.

Izjava o avtorstvu diplomskega dela

Spodaj podpisani Jurij Slabanja, z vpisno številko **63110281**, sem avtor diplomskega dela z naslovom:

Stabilizacija kvadrokopterja z vizualnim sledenjem oznaki

S svojim podpisom zagotavljam, da:

- sem diplomsko delo izdelal samostojno pod mentorstvom doc. dr. Danijela Skočaja,
- so elektronska oblika diplomskega dela, naslov (slov., angl.), povzetek (slov., angl.) ter ključne besede (slov., angl.) identični s tiskano obliko diplomskega dela,
- soglašam z javno objavo elektronske oblike diplomskega dela na svetovnem spletu preko univerzitetnega spletnega arhiva.

V Ljubljani, dne 17. septembra 2014 Podpis avtorja:

Kazalo

Povzetek

V tem delu smo implementirali postopek, s katerim lahko kvadrokopter Parrot AR.Drone sledi premikajočemu se objektu. Temelji na razvojnem okolju ROS. Uporabili smo mobilno platformo iRobot Roombo, nanjo nalepili oznako in jo vozili po prostoru, kvadrokopter pa se je moral ves čas držati čim bolj točno nad njo. Kvadrokopter je oznako zaznaval s pomočjo sprednje kamere, krmiljenje pa je bilo izvedeno s krmilnikom PID. Nalogo smo ovrednotili s tremi različnimi eksperimenti. Najprej smo ovrednotili kako natančna je naša metoda ocenjevanja višine kvadrokopterja v primerjavi s kvadrokopterjevim privzetim načinom. Nato smo ovrednotili kako natančno in robustno se zna kvadrokopter držati nad mirujočo oznako. Nazadnje smo ovrednotili še natančnost in robustnost pri premikajoči se oznaki.

Rezultati kažejo, da lahko z uporabo sistema implementiranega v tej nalogi kvadrokopter precej bolj stabilno lebdi nad oznako in ocenjuje razdaljo do nje, v primerjavi s privzetimi metodami, ki so vgrajene v Parrot AR.Drone.

Ključne besede: kvadrokopter, ROS, oznaka, kalibracija kamere, sledenje robov, Ramer–Douglas–Peucker algoritem, krmilnik PID.

Abstract

In this thesis we implemented a procedure, which Parrot AR.Drone can use to track a moving target. It is based on the ROS framework. We used an iRobot Roomba mobile platform as the target, taped a marker on top of it, and drove the Roomba around. The drone's goal was to stay as directly as possible above the target. The drone tracked the marker using its front camera and a PID controller was implemented to control the drone's movements. We conducted three different experiments to evaluate the implemented algorithms. First we evaluated how accurate our method of determining the drone's height was, compared to the drone's default method. Then we evaluated how accurately and robustly the drone can hover over a stationary target. Lastly, we evaluated the accuracy and robustness of the drone hovering over a moving target.

The results show that the system implemented in this thesis allows the drone to hover much more reliably over the marker, compared to AR.Drone's default stabilization methods. The system also provides a better means of estimating the distance of the drone from the marker, compared to the built-in sonar.

Keywords: quadcopter, ROS, marker, camera calibration, edge following, Ramer–Douglas–Peucker algorithm, PID controller.

Poglavje 1

Uvod

Zračna vozila brez posadke ali UAV (ang. Unmanned Aerial Vehicle) so pomembna, ker omogočajo ljudem priti na mesta, kamor sami ne morejo zaradi nevarnih okoliščin ali kakšnega drugega razloga. Ker imajo taka vozila povečini nameščene tudi kamere, jih je možno sprogramirati tako, da s pomočjo računalniškega vida samostojno letijo po prostoru. Primer take funkcionalnosti je sledenje objektom na tleh. V diplomski nalogi bomo razvili algoritem, s katerim bo kvadrokopter Parrot AR.Drone lahko sledil mobilni platformi iRobot Roomba, ki bo označena s preprosto oznako, kot prikazuje slika 1.3. V tem poglavju naprej obrazložimo motivacijo za to nalogo, nato pa njene cilje. Nazadnje predstavimo še oris naloge in zgradbo tega poročila.

1.1 Motivacija

Zračna vozila brez posadke, tudi znana kot daljinsko vodena zračna vozila ali RPA (ang. Remotely Piloted Aircraft), so vozila, ki za letenje ne potrebujejo človeške posadke. V preteklosti so potrebovala človeškega pilota, ki je s pomočjo daljinskega upravljalnika krmilil UAV, dandanes pa se vedno bolj uveljavlja avtonomno letenje. Le-to je izvedeno s pomočjo krmilnikov nameščenih na samem vozilu. Za take krmilnike lahko napišemo program, s katerimi lahko

UAV samostojno leti brez nadaljne človeške pomoči. Pogosto so se uporabljali za vojaške operacije, vedno bolj pa se uveljavljajo tudi v civilnih aplikacijah, npr. za policijsko in gasilsko delo, varnost in nadzor ipd. Načeloma velja, da se UAV uporablja, če je naloga preveč "nezanimiva, umazana ali nevarna" za navadno zračno vozilo s posadko.

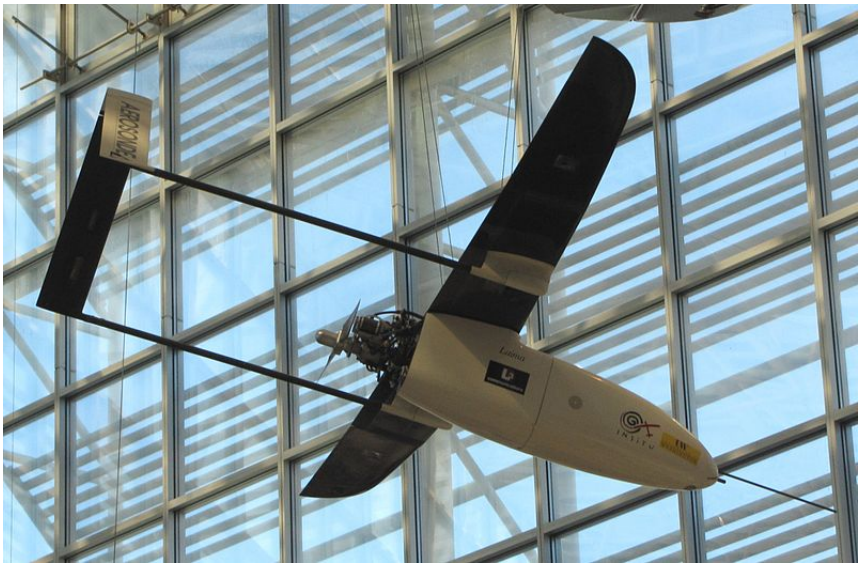
Glavna civilna področja, kjer se uporabljajo UAV so zaznavanje na daljavo, pomoč ob nesrečah, znanstveno raziskovanje, nadzor in zajemanje videa iz zraka.

Na UAV lahko namestimo mnogo različnih senzorjev. Z njimi lahko od daleč zaznavamo okolico v kateri se UAV nahaja. Z elektromagnetnimi senzorji lahko vidimo njegovo okolico. Biološki senzorji omogočajo zaznavanje mikrobov in drugih bioloških faktorjev v zraku. S kemičnimi senzorji pa lahko analiziramo sestavo elementov v zraku.

Vozila UAV lahko pomagajo tudi v primeru nesreč. Zmožni so prinašati zdravila in hrano in odnašati medicinske vzorce iz območij nesreč [8]. V takih primerih lahko pomagajo tudi tako, da zbirajo podatke o prizadetem območju.

Prav tako so pomembni za znanstveno raziskovanje. Leta 2006 je NOAA (ang. National Oceanic and Atmospheric Administration) začela uporabljati Aerosonde UAV za analizo orkanov [5]. Primer takega UAV je prikazan na sliki 1.1. Taka območja so preveč nevarna, da bi vanje pošiljali vozila s človeško posadko, z uporabo UAV pa lahko analizirajo orkan brez večjih nevarnosti. Za raziskovanje se UAV uporabljajo tudi v ekstremnih podnebjih, npr. obstajajo vozila narejena za uporabo na Antarktiki, ki lahko zdržijo izjemno nizke temperature.

Avtonomno zasledovanje objektov spada predvsem pod nadzor, delno pa tudi v zajemanje videa iz zraka. Pomembno je na mnogo področjih, med njimi so npr. filmska industrija, varnost in nadzor, snemanje športov ali drugih dejavnosti na prostem ipd. Z uporabo UAV lahko na preprost način zajamemo slike ali video iz zraka. Če na tako vozilo naložimo program, s katerm bo znal sam slediti igralcem v filmu ali tekmovalcem v dirki, lahko zelo poenostavimo



Slika 1.1: UAV Aerosonde Laima, uporabljen za preučevanje orkanov. Vir: [17].

delo snemalcem.

Taka vozila so tudi zelo uporabna v primerih, ko bi bilo nevarno poslati človeka na teren. Na primer obstajajo algoritmi, ki lahko spremljajo širjenje požara [14]. Če tak program naložimo na kvadrokopter in ga pošljemo nad požar, ne ogrožamo človeških življenj in lahko enostavno spremljamo, kako se požar širi v realnem času. Primer takega kvadrokopterja lahko vidimo na sliki 1.2.

1.2 Cilj diplomske naloge

Cilj diplomske naloge je razviti program, s katerim bo kvadrokopter Parrot AR.Drone zmožen slediti premikajočemu se objektu. V našem primeru bo to mobilna platforma iRobot Roomba, ki bo označena s preprosto oznako. Roombo bomo lahko nato z igralnim ploščkom ali preko tipkovnice krmilili skozi prostor. Kvadrokopter bo moral, s pomočjo navzdol obrnjene kamere, ves čas lebdeti čim bolj točno nad oznako oz. Roombo. V primeru da jo

Slika 1.2: Kvadrokopter uporabljen pri opazovanju požarov. Vir: [4].

izgubi, je zaželjeno, da jo zna tudi sam spet najti.

Program bo napisan s pomočjo razvojnega okolja ROS, v jeziku C++. Izvorna koda bo na voljo tu [11].

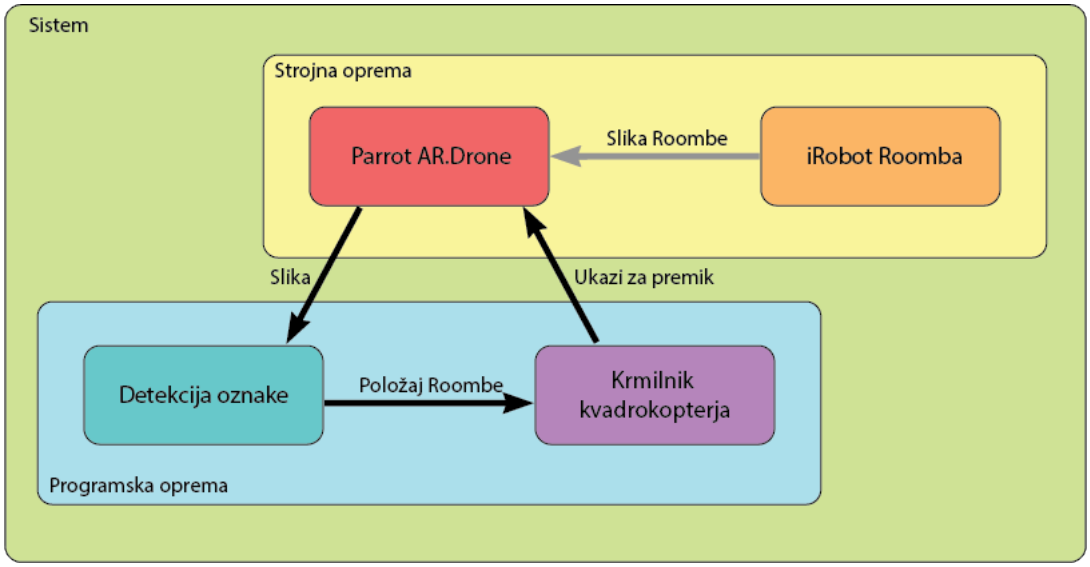
1.3 Oris sistema

Razviti sistem lahko v grobem razdelimo na dva sklopa, strojno in programsko opremo. Vsakega od teh dveh sklopov lahko nato še naprej delimo na dva podsklopa. Pod strojno opremo spadata kvadrokopter Parrot AR.Drone in mobilna platforma iRobot Roomba. V sklopu programske opreme pa smo v nalogi implementirali detekcijo oznake in krmiljenje kvadrokopterja.

Ti sklopi so medseboj povezani, kot kaže slika 1.4. Povezava med Roombo in kvadrokopterjem je šibkejša, ker kvadrokopter pravzaprav le zajema slike ni pa nobene druge komunikacije med njima. Kvadrokopter nato te slike pošilja algoritmu za detekcijo oznake. Le-ta iz slik ugotovi, kje v prostoru, relativno

Slika 1.3: Kvadrokopter Parrot AR.Drone, ki lebdi nad iRobot Roombo.

nanj, se Roomba nahaja in te koordinate pošlje krmilniku. Krmilnik na podlagi koordinat pošlje ustrezne ukaze za premik kvadrokopterju in ta postopek se potem ponavlja.



Slika 1.4: Grafični prikaz sklopov diplomske naloge.

1.4 Zgradba diplomske naloge

Naloga je razdeljena na šest poglavij. V poglavju 1 povemo nekaj uvodnih besed, motivacijo, cilje, oris in zgradbo diplomske naloge.

V poglavju 2 opišemo že obstoječe sisteme, ki smo jih uporabili v nalogi. Podamo tehnične sprecifikacije kvadrokopterja Parrot AR.Drone in na kratko opišemo razvojno okolje ROS.

Poglavji 3 in 4 opisujeta naš prispevek, kaj smo implementirali v nalogi in kako smo to naredili. V poglavju 3 opišemo zajemanje slik, umerjanje kamere, zaznavanje oznake in preslikavo med koordinatnimi sistemi. Poglavje 4 opisuje, kako smo realizirali krmiljenje kvadrokopterja.

V poglavju 5 predstavimo rezultate testiranja naloge.

Nazadnje pa v poglavju 6 povzamemo vse ugotovitve v sklep in predlagamo nekaj področij, kjer bi lahko sistem izboljšali.

Poglavje 2

Mobilna platforma Parrot AR.Drone

Za izvedbo te diplomske naloge sta bila integralnega pomena dva sistema, kvadrokopter Parrot AR.Drone in razvojno okolje ROS. V tem poglavju bomo naprej opisali kvadrokopter in njegove tehnične specifikacije, nato pa bomo še na kratko predstavili ROS.

2.1 Tehnične specifikacije

Parrot AR.Drone je kvadrokopter, ki ga je zasnovalo francosko podjetje Parrot [6][18]. Upravljamo ga lahko preko brezžičnega omrežja Wi-Fi. V osnovi je bil namenjen uporabi s pametnimi telefoni, vendar dandanes obstajajo tudi knjižnice in razvojna okolja, s katerimi ga lahko krmilimo tudi s pomočjo računalnika. Trenutno sta na trgu dve različici, AR.Drone 1.0, narejen leta 2010 in AR.Drone 2.0, narejen leta 2012. V tej nalogi smo uporabili različico 2.0. Le-ta je bila nadgradnja prve in je vsebovala mnoge izboljšave, tako za programsko kot tudi za strojno opremo. Glavne razlike so izboljšana kamera in senzorji, kar omogoča mnogo boljši nadzor, dodan senzor za zaznavanje zračnega tlaka ter posodobljen standard za Wi-Fi. Prav tako so nadgradili

žiroskop, pospeškomer in magnetomer v tri-osne.

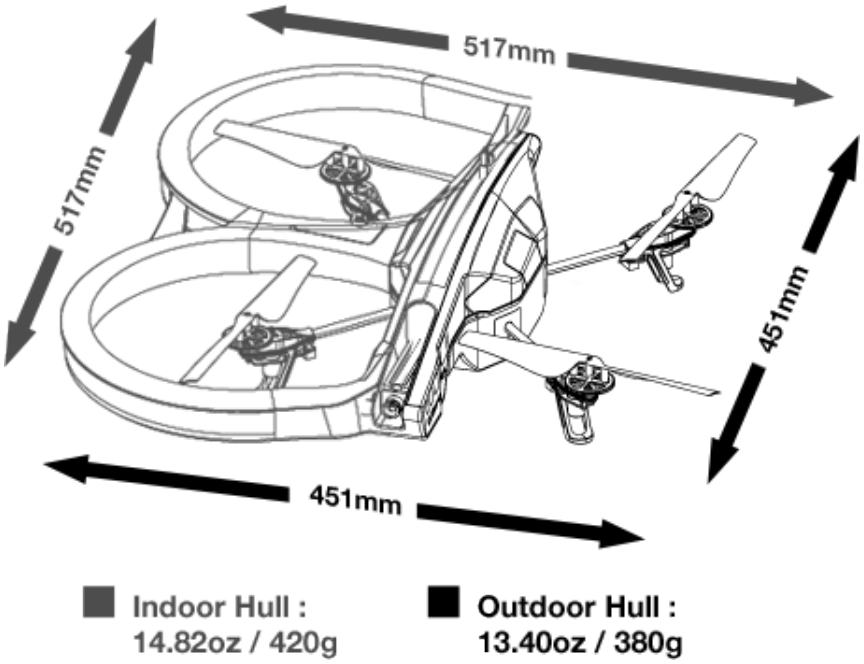
Kvadrokopter ima dve kameri. Sprednja kamera ima ločljivost 720p in objektiv z zornim kotom velikim 92°, podpira pa snemanje s hitrostjo do 30 sličic na sekundo ali fps (ang. frames per second). Spodnja je nekoliko slabša. Je senzor formata QVGA z zornim kotom velikim 64°, vendar lahko snema do 60 fps. To kamero kvadrokopter tudi uporablja za stabilizacijo. Če so tla dovolj teksturirana, bi kvadrokopter s spodnjo kamero moral zaznati to podlago in lebdeti na približno istem mestu. Poleg žiroskopa, pospeškomera, magnetomera, senzorja za tlak in kamer, so na kvadrokopterju nameščeni še ultrazvočni senzor za ocenjevanje višine in senzor za temperaturo, prav tako pa zna tudi približno oceniti kako močno in v katero smer piha veter. Procesor je 1GHz 32 bit ARM Cortex A8, na njem pa teče operacijski sistem Linux 2.6.32. Ima tudi 1Gbit DDR2 RAM. Shema kvadrokopterja je prikazana na sliki 2.1. Sliki 2.2 prikazujeta kvadrokopter, ki je bil uporabljen v diplomski nalogi.

2.2 Razvojno okolje ROS

V splošnem je robustno programiranje poljubnega robota dokaj zahtevna naloga. Iz tega razloga je bil razvit ROS (ang. Robot Operating System). ROS je prilagodljivo okolje za pisanje programske opreme za robote. Je skupek orodij, standardov in knjižnic oz. paketov, ki poenostavijo programiranje robustne programske opreme, za mnogo različnih robotskih platform. V osnovi deluje sistem kot oglasna deska, razdeljena na več tem (ang. topic). Program lahko napišemo kot vozlišče ali več vozlišč, ki objavlja ali bere sporočila iz določene teme in na tak način posamezna vozlišča komunicirajo med sabo.

Osnovna funkcionalnost [9] sistema ROS je že omenjena infrastruktura za komunikacijo. Ima pa tudi implementirane specifične funkcije za upravljanje z roboti.

Pogosto je težko spremljati kje se v vsakem trenutku vsak del robota na-



Slika 2.1: Shema Parrot AR.Drona, na levi z ohišjem za zaprte prostore, na desni z ohišjem za letenje na prostem. Vir: [6].

haja. Zato ROS nudi paket tf, ki poenostavi ta problem in omogoča preprosto pretvarjanje med različnimi koordinatnimi sistemi. Omogoča tudi opis robota v formatu URDF (ang. Unified Robot Description Format). Na tak način lahko opišemo fizične lastnosti robota, npr. dolžina rok, velikost koles ipd. Ti podatki se nato uporabijo za pretvarjanje med koordinatnimi sistemi in vizualizacijo robota.

ROS vsebuje tudi pakete, ki poenostavijo osnovne probleme v robotiki, npr. ocenjevanje drže robota, lokalizacija in gradnja zemljevida. Na tak način ROS omogoči uporabnikom, da relativno hitro naredijo več z manj truda.

Te funkcije delujejo na poljubnih robotih, ker infrastruktura za komunikacijo zagotavlja tudi standardne formate za sporočila. Ti formati pokrivajo večino najpogosteje uporabljenih podatkov v robotiki. To so npr. podatki za geometrijske koncepte (npr. drža robota in vektorji), za senzorna sporočila

Slika 2.2: Kvadrokopter Parrot AR.Drone 2.0 uporabljen v diplomski nalogi.

(npr. kamera in laser) in za navigacijske podatke (npr. odometrija in zemljevidi).

Vse funkcionalnosti sistema ROS je možno poganjati neposredno iz ukazne vrstice. Poleg tega nudi tudi dva grafična vmesnika, rqt in rviz. Oba imata poudarek predvsem na vizualizaciji. Med njima je bolj znan rviz. Ponuja vizualizacijo mnogo pogosto uporabljenih sporočilnih formatov, npr. tridimenzionalni točkovni oblak, zemljevid, vizualizacija robota in njegove drže in še mnogo več. Z njegovo pomočjo lahko vidimo kako robot zaznava okolico in hitro ugotovimo kaj je šlo med izvajanjem določenega programa narobe.

Mnogo robotskih platform ima s pomočjo paketov ROS napisane krmilnike ali pa vmesnik za krmilnike. Le-ti spremljajo določene teme in sporočila iz njih prevedejo v ukaze, ki jih pošljejo robotu, da le-ta nekaj naredi. Za Parrot AR.Drone obstaja nekaj takih paketov, v tej nalogi smo uporabili ardrone_autonomy [2], ki podpira obe verziji AR.Drona in večino senzorjev na njem. Zgrajen je na osnovi ArdroneSDK, ki zagotavlja uradne krmilnike za kvadrokopter. Glavni temi, ki jih objavlja, sta *navdata* (t.j. podatki iz vseh senzorjev) in slika iz kamere. Kvadrokopterju lahko pošiljamo ukaze prek štirih tem - ena za vzlet, ena za pristanek, ena za krmiljenje in ena za ponastavitev kvadrokopterja. ardrone_autonomy prebere ukaze iz teh tem in jih pretvori v ukaze ArdroneSDK, s katerimi nato krmili kvadrokopter.

Poglavje 3

Zaznavanje oznake

Praktični del naloge smo razdelili na dva glavna dela. Zaznavanje oznake in krmiljenje kvadrokopterja. V tem poglavju malo podrobneje predstavimo, kako je bilo izvedeno zajemanje slik, kalibracija kamere, zaznavanje oznake in pretvorba iz slikovnega v globalni koordinatni sistem. Na koncu še na kratko predstavimo nekaj implemetacijskih podrobnosti za zaznavanje oznake.

3.1 Zajemanje slik

Oznako je kvadrokopter iskal s pomočjo navzdol obrnjene sprednje kamere. Za to ga je bilo potrebno nekoliko modificirati, saj privzeto sprednja kamera kaže naravnost naprej. Kako smo to naredili, lahko vidimo na sliki 3.1. Za sprednjo kamero smo se odločili, ker ima širši zorni kot in kvadrokopter tako manjkrat izgubi oznako iz svojega vidnega polja. Iz istega razloga smo poskusili zagotoviti, da se kvadrokopter med letom drži dovolj visoko nad oznako. Algoritem za krmiljenje ga poskuša držati na višini 1.5 metra.

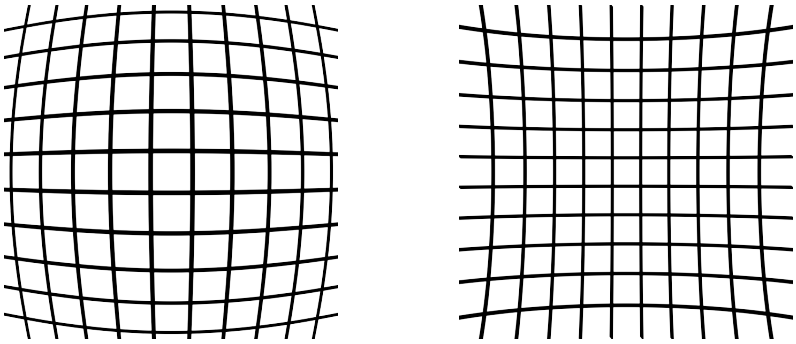
Sprednji objektiv tudi ukrivi sliko bolj kot spodnji. Zato je bilo potrebno kamero umeriti, kot je opisano v poglavju 3.2 in odstraniti popačenje slike.

(a) Privzeti položaj (b) Spremenjen položaj

Slika 3.1: Modifikacija sprednje kamere.

3.2 Kalibracija kamere

Sprednja kamera kvadrokopterja ima širokokotni objektiv, saj ima zorni kot velik 92°. S takimi objektivi pogosto pride do pojava, ki ga imenujemo radialno popačenje. Primere takega popačenja lahko vidimo na slikah 3.2.

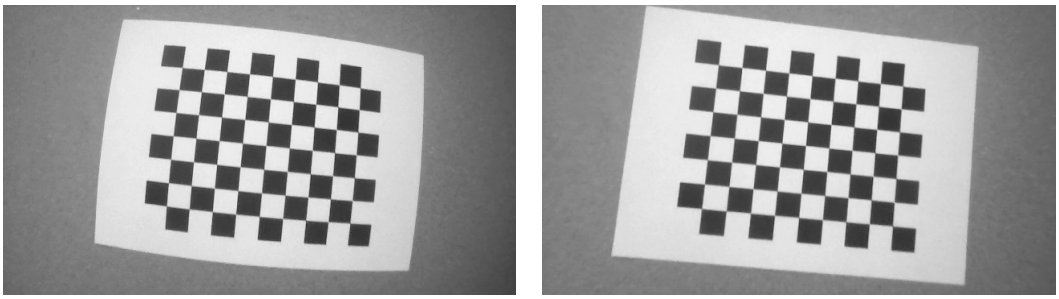


(a) Sodčasto (b) Blazinasto

Slika 3.2: Različne vrste radialnega popačenja.

Taka ukrivljenost slike ni zaželjena, saj zaplete preslikavo oznake iz slikovnih koordinat v koordinate sveta. Lažje je najprej popraviti sliko in šele nato izvesti preslikavo. Da lahko to storimo, moramo poznati notranje parametre kamere. Le-te dobimo tako, da kamero umerimo. Pri tem dobimo koeficiente popačenja, s katerimi lahko popravimo sliko. Primer odstranitve popačenja

lahko vidimo na sliki 3.3.



(a) Popačena slika (b) Popravljena slika

Slika 3.3: Popravek popačene slike.

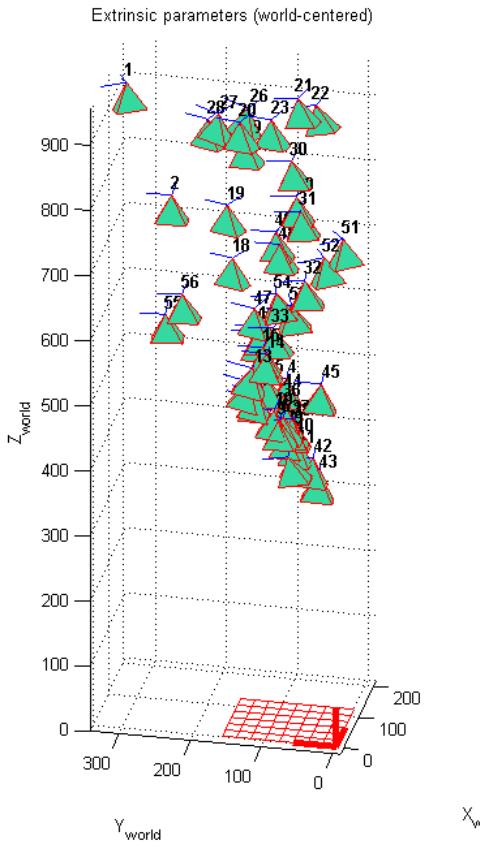
Umerjanje smo izvedli tako, da smo s kamero zabeležili slike šahovnice iz različnih pozicij, kar kažejo slike 3.4 in 3.5. Ker vemo, da bi morala presečišča šahovnice ležati na vzporednih premicah, lahko na tak način iz presečišč določimo ukrivljenost slike. Ukrivljenost določimo za vse zabeležene slike in izračunamo povprečje za vsak parameter.

Vrednosti, ki smo jih na tak način dobili za notranje parametre sprednje kamere kvadrokopterja, so podane v tabeli 3.1.

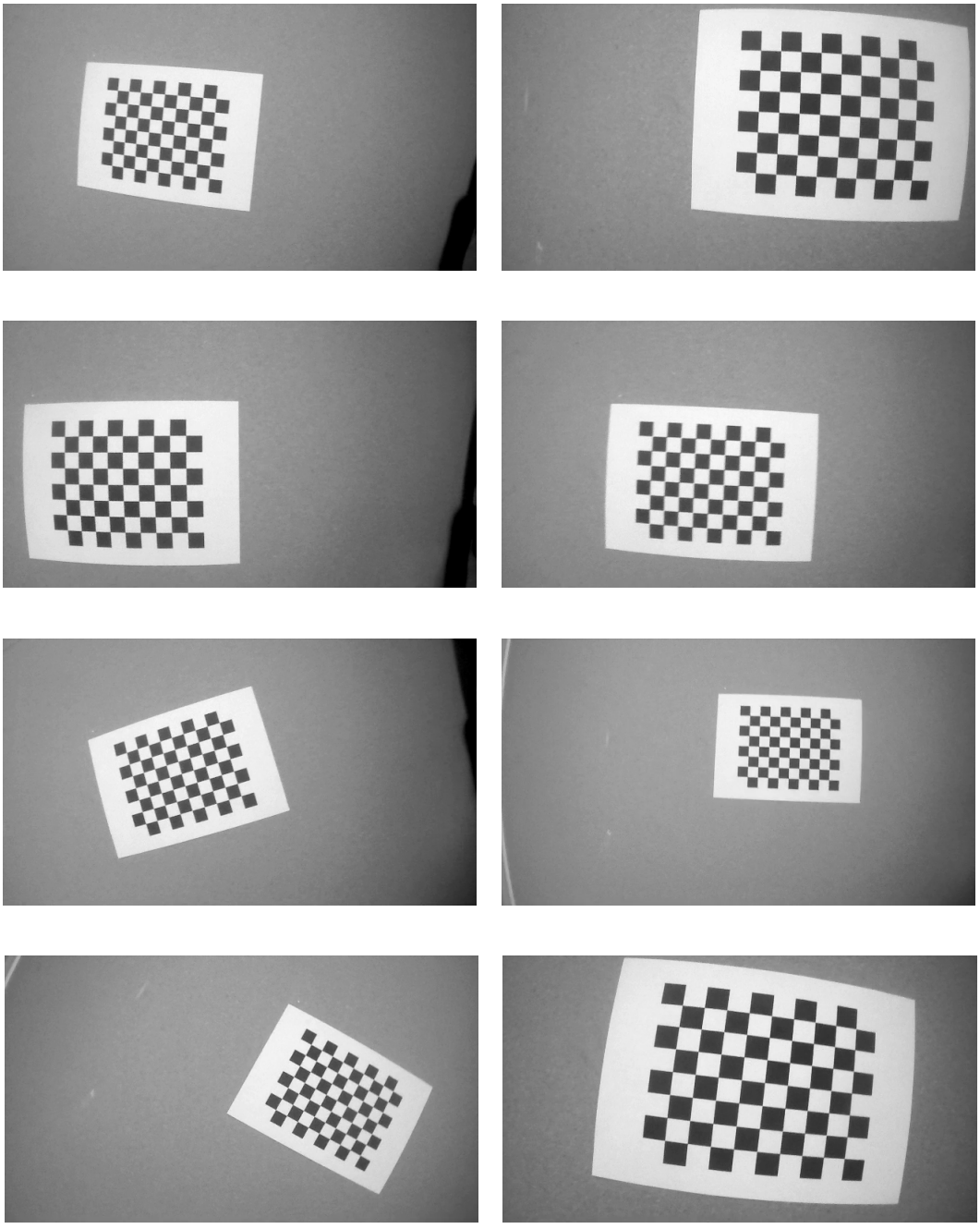
Slika 3.4 prikazuje položaje kamere, od koder smo zajeli slike za kalibracijo. Zeleni trikotniki prikazujejo položaj kamere, rdeča mreža na dnu pa položaj šahovnice. Opazimo lahko, da smo slike za umerjanje kamere zajeli iz precej omejenega območja. V splošnem bi bilo bolje, če bi nekaj slik zajeli še bolj od strani in s tem dobili bolj natančne vrednosti za parametre kamere. V našem primeru tako prekomerno ustrezanje (ang. overfitting) določenemu pogledu ni neprimerno, saj bo kvadrokopter vedno zaznal oznako iz podobnega pogleda.

Širina slike [px]	640
Višina slike [px]	360
Goriščna razdalja fx [px]	595.51
Goriščna razdalja fy [px]	595.61
Glavna točka cx [px]	330.32
Glavna točka cy [px]	160.33
Popačenje $k1$	-0.63109
Popačenje $k2$	0.65711
Popačenje $k3$	0.00656
Popačenje $k4$	-0.00155
Popačenje $k5$	-0.53136
Povprečna reprojekcijska napaka	0.16330

Tabela 3.1: Parametri sprednje kamere na AR.Drone 2.0



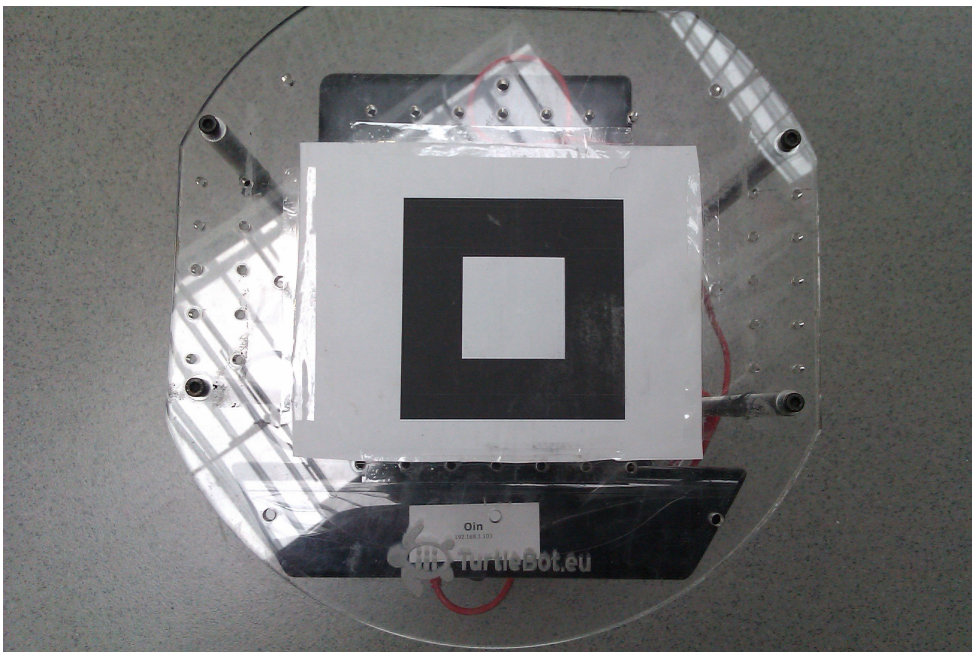
Slika 3.4: Položaji kamere glede na kalibracijsko šahovnico.



Slika 3.5: Primeri slik uporabljenih za kalibracijo kamere.

3.3 Detekcija oznake

Oznaka je preprosta, kot lahko vidimo na sliki 3.6. Je manjši beli kvadrat na večjem črnem kvadratu s skupnim središčem. Tri glavne faze algoritma lahko tudi vidimo na slikah 3.7. Algoritem deluje tako, da najprej sliko pretvori v sivinsko. To je prikazano v levem delu slik 3.7. Nato sliko binarizira. To je prikazano v srednjem delu slik. 3.7. Nazadnje iz slike izloči kvadrate in določi kje na sliki se nahaja oznaka. To je prikazano v desnem delu slik 3.7. Bolj podroben potek prikazuje Algoritem 1.



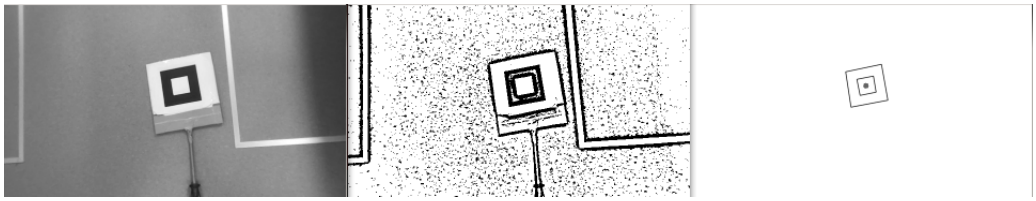
Slika 3.6: Oznaka, kateri je moral kvadrokopter slediti.

Največji problem z algoritmom je, da višje kot se kvadrokopter povzpne, težje razloči med notranjim in zunanjim kvadratom oznake. Na določeni višini črna obroba postane tako tanka, da jo zazna kot eno samo črto. Iz tega razloga gleda ploščine kvadratov, če ne more določiti oznake na podlagi skupnega središča. Pri tem tudi dosti lažje pride do napačnih zaznav, vendar se izkaže, da v praksi dovolj dobro dela, če tla niso kvadratno teksturirana.

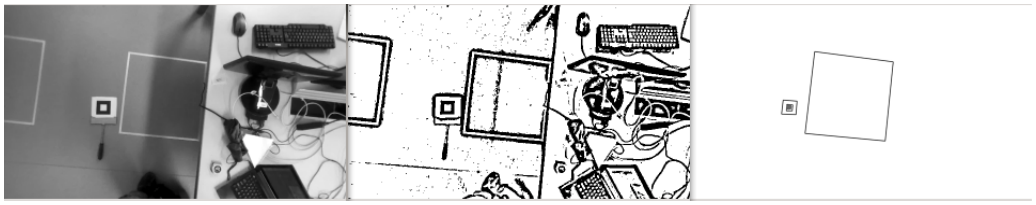
Data: Slika iz sprednje kamere
Result: Pozicija oznake v prostoru, relativno na kvadrokopter

```
Spremeni sliko v sivinsko predstavitev;
Popravi ukrivljenost slike;
Binariziraj sliko z adaptivnim gaussovim pragom;
Na sliki določi vse like;
foreach lik na sliki do
    Poenostavi lik (zmanjšaj število točk ki ga definira);
    if lik je konveksen štirikotnik in ni izrazito majhen then
        if štirikotnik ima vse stranice približno enako dolge in kote 90° then
            Na sliki smo našli kvadrat;
        end
    end
end
foreach kvadrat na sliki do
    Izračunaj središče kvadrata;
    if središče se prekriva s katerim prejšnjim središčem then
        Našli smo oznako;
    end
end
if še nismo našli oznake then
    foreach kvadrat na sliki do
        if ima pravo ploščino then
            Našli smo oznako;
        end
    end
end
Pretvori položaj oznake na sliki v položaj oznake v prostoru;
```

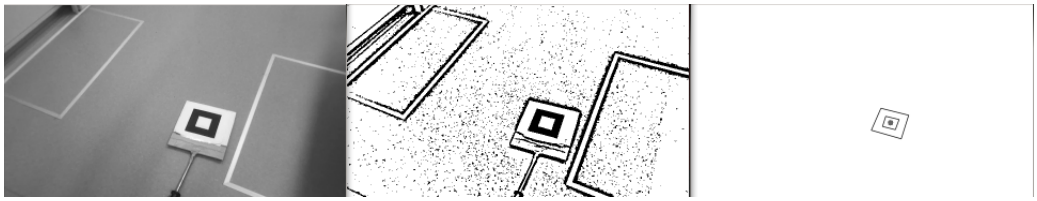
Algorithm 1: Algoritem za zaznavanje oznake.



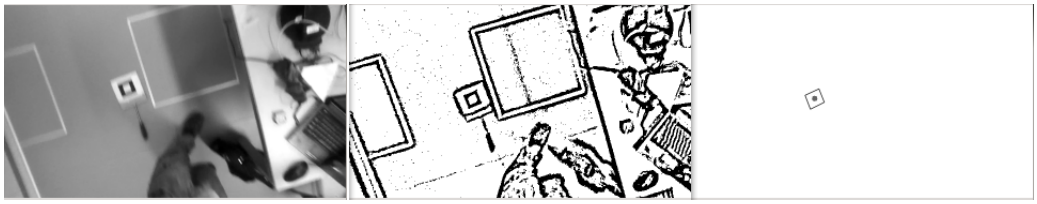
(a) Zaznana oznaka. Krog označuje središče oznake.



(b) Poleg oznake je bil na sliki zaznan še dodatni kvadrat.



(c) Algoritem je precej robusten, tudi pri večjih nagibih kvadrokopterja.



(d) Oznaka zaznana samo na podlagi ploščine kvadrata.

Slika 3.7: Tri glavne faze obdelave slik - sivinska slika, binarizacija in določanje kvadratov na sliki.

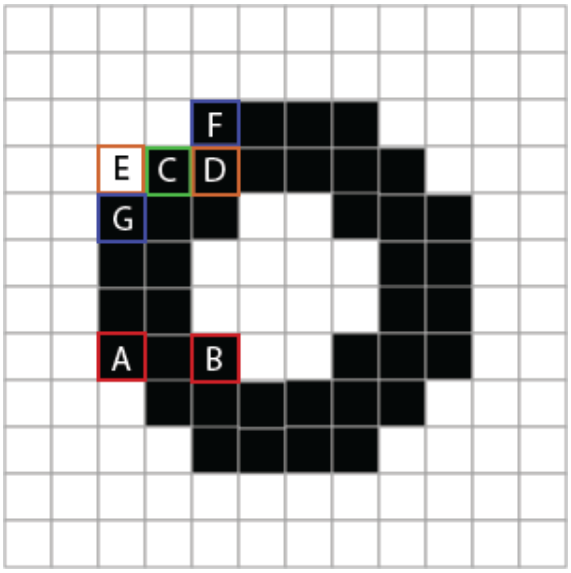
Zaznavanje obrob deluje na principu rasterskega skeniranja in sledenja robov [12]. Najprej definirajmo kaj pomeni, da je neka točka na sliki del roba lika. Točka (i, j) je del zunanjega roba, če je (i, j) črna točka in $(i, j-1)$ bela. Za notranji rob pa mora veljati, da je (i, j) črna točka in $(i, j+1)$ bela. Zunanji rob označuje obris lika, na sliki 3.8 je točka zunanjega roba ponazorjena s črko A. Notranji rob označuje obris luknje, na sliki 3.8 je točka notranjega roba

ponazorjena s črko B.

Z rasterskim skeniranjem se sliko pregleda po en slikovni element (točko) naenkrat, od leve proti desni, od zgoraj navzdol. Za vsako točko se ugotovi ali je del zunanjega ali notranjega roba. Če ni del roba se skeniranje nadaljuje. Če je, se pa rob označi z zaporedno številko in se mu začne slediti. Sledenje je izvedeno tako, da se pogleda sosede trenutne točke (i, j). Najde se prvega soseda (i_1, j_1) v smeri urinega kazalca, ki zadostuje pogoju za rob. Nato se najde prvega soseda (i_2, j_2) v nasprotni smeri urinega kazalca, ki zadostuje pogoju za rob. Na tak način se določi tri zaporedne točke na robu. Točko (i, j) se označi z zaporedno številko roba. Nato se algoritem premakne iz točke (i, j) v točko (i_1, j_1) ter iz (i_2, j_2) v (i, j). Če se je premaknil na že označeno točko pomeni, da je zaključil s sledenjem robu in lahko nadaljuje z rasterskim skeniranjem slike. Sicer nadaljuje s sledenjem robu.

Za boljšo ponazoritev omenjenih pojmov si poglejmo primer na sliki 3.8. Črne točke predstavljajo okrogel lik z okroglo luknjo na sredini. Bele točke predstavljajo ozadje slike. C označuje točko (i, j), D označuje $(i, j + 1)$ in E označuje $(i, j - 1)$. F predstavlja točko (i_1, j_1), G pa točko (i_2, j_2). V tem primeru je algoritem začel rastersko skenirati v četrti vrstici in se ustavil, ko je prišel do točke C, ki zadostuje pogoju za zunanji rob. V naslednji iteraciji bo algoritem začel slediti robu, tako da bo C označil z zaporedno številko in se premaknil iz C v F ter iz G v C, nato pa poiskal naslednjo točko na zunanjem robu.

Ko so na sliki določene vse obrobe likov, je pametno, če se jih poenostavi. Tako se ne operira več z npr. tisočimi točkami na en štirikotnik, ampak samo še s štirimi. Algoritem, ki to izvede, se imenuje Ramer-Douglas-Peucker algoritem [7]. Ideja algoritma je, da če imamo podano krivuljo, sestavljeno iz več ravnih segmentov, se za to krivuljo najde podobno krivuljo, sestavljeno iz manj segmentov. Le-ta od originalne ne sme odstopati za več kot ϵ. Algoritem ne ustvarja novih točk, le odstrani tiste, ki so odveč. Najprej najde dve točki A in B. Na začetku sta inicializirani na začetno in končno točko krivulje. A in



Slika 3.8: Zaznavanje obrob s pomočjo sledenja robov.

B označi, da sta dobri. Dobre točke bo algoritem vedno obdržal. Nato poišče točko C, ki je od premice $\overline{AB}$ najbolj oddaljena. Ta točka je očitno najdlje stran od pravilne aproksimacije krivulje. Če je razdalja med C in $\overline{AB}$ manjša od ϵ, potem se lahko zavrže vse točke med A in B brez strahu, da bi bila krivulja slabša od ϵ. Postopek se nato rekurzivno ponavlja s točkama A in C ter točkama C in B. Ko zaključi, dobimo krivuljo sestavljeno samo iz dobrih točk, ki od originalne krivulje ne odstopa za več kot ϵ.

3.4 Preslikava koordinatnega sistema

Pretvorba položaja iz slikovnih koordinat v globalne temelji na podobnih trikotnikih, kot je prikazano na sliki 3.9. Za podobne trikotnike velja

$$\frac{x'}{h+f} = \frac{x}{f}, \tag{3.1}$$

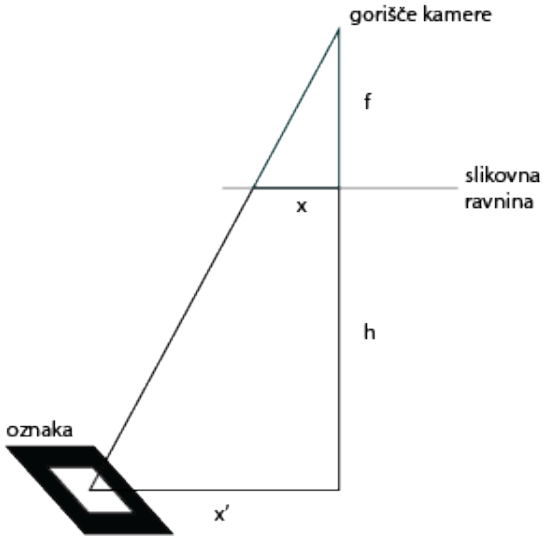
tako da lahko formulo za pretvorbo izrazimo kot

$$x' = \frac{x(h+f)}{f}, \tag{3.2}$$

kjer je x' položaj v globalnih koordinatah, x položaj na sliki, f goriščna razdalja kamere in h oddaljenost kamere od oznake. Da lahko uporabimo enačbo, moramo najprej vse vrednosti pretvoriti v iste merske enote. Vse, razen položaja na sliki, imamo podano v metrih, le-ta pa je podan v slikovnih elementih (ang. pixel). Pretvorimo ga v metre tako, da ga množimo z gostoto slikovnih elementov kamere. To gostoto lahko izraunčamo kot

$$g = \frac{w_{senzor}}{n_{vrstica}}, \tag{3.3}$$

kjer je g gostota slikovnih elementov, w_{senzor} širina senzorja kamere v metrih in $n_{vrstica}$ število slikovnih elementov na posamezno vrstico senzorja.



Slika 3.9: Izračun globalne pozicije oznake iz položaja na sliki.

Kvadrokopter ima za zaznavanje višine vgrajen ultrazvočni senzor, vendar je za naše potrebe preveč nezanesljiv. Ker je oznaka nalepljena na vrhu mobilne platforme iRobot Roomba, senzor ne meri vedno razdalje od kvadrokopterja do oznake. To se zgodi samo, če kvadrokopter lebdi točno nad oznako. Sicer meri razdaljo od kvadrokopterja do tal. Tako nihanje v vrednostih lahko zmanjša natančnost pri ocenjevanju položaja oznake. Ultrazvočni senzor tudi ni najbolj natančen, kar pokažemo v poglavju 5. Z uporabo samo tega senzorja se je

mnogokrat zgodilo, da je kvadrokopter izgubil višino in se ni več dvignil nazaj na pravo mesto. To je bilo še posebej očitno vsakič, ko so se mu baterije nekoliko spraznile. Zato smo implementirali ocenjevanje višine na podlagi oznake. Ker poznamo dimenzije oznake in notranje parametre kamere, lahko trenutno višino zračunamo kot

$$h = \frac{w_{oznaka\_realna} * f}{w_{oznaka\_na\_sliki}} - f, \tag{3.4}$$

kjer je h višina, f goriščna razdalja kamere in w označuje širino oznake. Pri tem je potrebno širino oznake na sliki pretvoriti iz slikovnih elementov v metre tako, da se $w_{oznaka\_na\_sliki}$ množi z gostoto slikovnih elementov kamere. Gostoto slikovnih elementov se lahko izračuna s pomočjo formule 3.3.

3.5 Implementacijske podrobnosti

Zaznavanje oznake sloni na knjižnici OpenCV. To je odprtokodna knjižnica namenjena realnočasovnemu reševanju problemov s področja računalniškega vida. Glavne točke programa vse kličejo OpenCV-jeve funkcije.

Ukrivljenost se popravi s funkcijo `undistort()`, na podlagi izmerjenih notranjih parametrov kamere. Le-te smo dobili z umerjanjem kamere. To smo naredili tako, da smo s kamero zabeležili slike šahovnice, nato pa te slike obdelali s pomočjo OpenCV-jeve skripte [15] za kalibracijo kamere. Nekoliko bolj podrobno je delovanje te skripte opisano v poglavju 3.2.

Za binarizacijo obstaja funkcija `adaptiveThreshold()`. Sliko spremeni iz sivinske v črno-belo tako, da namesto enega praga za celo sliko, izračuna več manjših lokalnih pragov za posamezne dele slike. Preverjanje konveksnosti likov je izvedeno z `isContourConvex()`.

Bolj zanimivi pa sta funkciji `findContours()` in `approxPolyDP()`, ki iz slike izločita obrobe likov in poenostavita te like na minimalno število točk. Kako delujeta ta dva algoritma je opisano v poglavju 3.3.

Poglavje 4

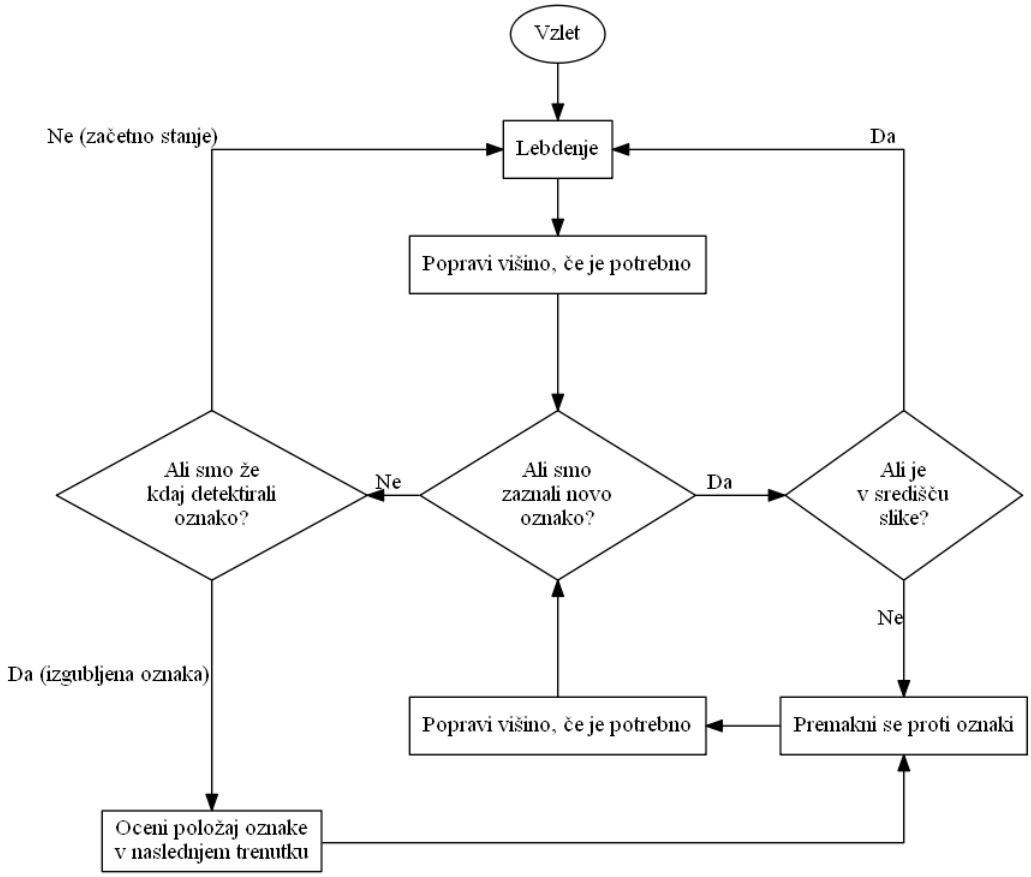
Krmiljenje kvadrokopterja

V tem poglavju predstavimo kako smo realizirali krmiljenje kvadrokopterja. Najprej na splošno opišemo algoritem. Nato predstavimo krmiljenje kvadrokopterja, ko ima le-ta oznako v svojem vidnem polju. Nazadnje še opišemo krmiljenje kvadrokopterja, ko oznako izgubi iz vidnega polja.

4.1 Osnova krmiljenja

Krmiljenje ima dve glavni nalogi. Držati mora kvadrokopter na čim bolj konstantni višini in čim bolj direktno nad oznako. Osnovna zanka algoritma za krmiljenje je razvidna iz grafa prikazanega na sliki 4.1. Izvedla se je približno desetkrat na sekundo. Osnovno stanje zanke je lebdenje kvadrokopterja na mestu. Vsako iteracijo se preveri, če je kvadrokopter zaznal oznako v svojem vidnem polju. V primeru da jo je, se kvadrokopter premankne proti njej. V nasprotnem primeru poskuša oceniti, kje se oznaka nahaja in se premakne v ustrezno smer. Zanka se je lahko prekinila s signalom SIGTSTP (Ctrl+Z) ali signalom SIGINT (Ctrl+X). Oba signala lahko pošljemo programu prek ukazne vrstice. SIGTSTP je ukazal kvadrokopterju naj pristane, SIGINT je pa povzročil zasilno ustavitev kvadrokopterja.

Da bi zagotovili čim bolj natančno zaznavanje oznake, se je kvadrokop-



Slika 4.1: Osnovna zanka za sledenje oznaki.

ter premikal počasi. Problem, ki nastane pri hitrih premikih je, da se mora kvadrokopter močno nagniti za tak premik in pri tem lahko zazna oznako v napačnem predelu slike. Poleg tega se je tudi poskušal držati dovolj visoko, da je lahko z manjšimi premiki hitreje prišel nazaj nad oznako. Tako je imel tudi večjo površino v svojem vidnem polju in je težje izgubil oznako.

Višino je popravljal samo, če je zaznal, da je prenizko. Lahko se je torej zgodilo, da je letel nekoliko višje, kot je imel določeno. Vendar se je v praksi povečini držal na pravi višini, ker se je dovolj počasi dvigal in se ni nikoli sam od sebe dvignil, le občasno se je spustil nekoliko nižje.

4.2 Krmilnik PID

Za premikanje proti oznaki smo implementirali dva krmilnika PID (ang. Proportional Integral Derivative controller) [10][3][1]. Enega za os x in drugega za os y oz. levo-desno in naprej-nazaj. PID je v osnovi kontrolna zanka s povratno informacijo in en najbolj uporabljenih krmilnikov v industriji. Shemo krmilnika PID lahko vidimo na sliki 4.2. Sestavljen je iz treh členov, proporcionalnega, integralskega in derivacijskega. Krmilnik računa odstopanje izmerjene spremenljivke v procesu od nastavljene vrednosti (ang. setpoint). V našem primeru to pomeni kako daleč stran od sredine slike po osi x oz. y smo zaznali oznako. Vsak člen meri drugačno odstopanje oz. napako. Proporcionalni meri trenutno napako, integralski preteklo napako in derivacijski napako, ki bo še nastala. Celoten krmilnik lahko izrazimo s formulo

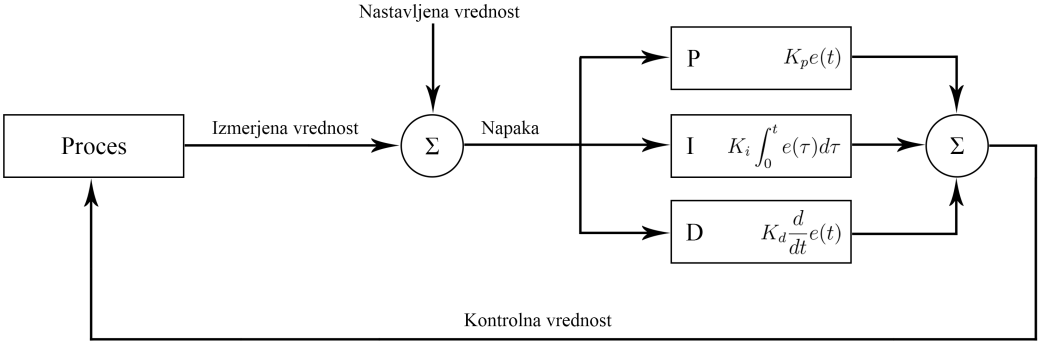
$$u(t) = K_p e(t) + K_i \int_0^t e(\tau)d\tau + K_d \frac{d}{dt} e(t), \tag{4.1}$$

kjer je K_p proporcionalni dobiček, K_i integralski dobiček, K_d derivacijski dobiček, e napaka, t trenutni čas in τ spremenljivka za integriranje, ki zavzame vrednosti 0 do t.

Proporcionalni člen kot izhod vrne vrednost, ki je sorazmerna s trenutno napako. Koliko ta člen vpliva na celotni izhod, lahko nastavimo z dobičkom K_p. Večji kot je K_p, močnejši bo odziv izhoda iz krmilnika. Tako lahko naredimo proces bolj odziven, vendar če nastavimo K_p previsoko, pogosto lahko pride do nenadzorovanega nihanja.

Integralski člen je sorazmeren z velikostjo in časom trajanja napake. Je pravzaprav vsota vseh napak do tega trenutka in poda nabrano napako, ki bi jo morali do tedaj popraviti. Tudi prispevek tega člena lahko uravnavamo z dobičkom. Če K_i nastavimo previsoko, ponavadi pride do precejšnje prekoračitve nastavljene vrednosti in širokega nihanja. Uporablja se za zmanjšanje napake v stabilnem stanju (ang. steady-state error). Le-ta nastane, ko proces neha nihati, vendar se ne umiri na nastavljeni vrednosti.

Derivacijski člen poskuša oceniti, kolikšna bo napaka v naslednjem trenutku, na podlagi naklona trenutne napake. Kot pri proporcionalnem in integralskem členu imamo tudi tu dobiček, s katerim lahko nastavljamo prispevek k celotnemu izhodu. Ker poskuša predvideti, kaj se bo zgodilo, ta člen poveča stabilnost procesa in zmanjša čas, ki ga proces rabi, da pride v stabilno stanje.



Slika 4.2: Shema krmilnika PID.

Najbolj zahtevna naloga je bila umeriti krmilnik PID tako, da je kvadrokopter čim manj nihal okoli oznake in je bil hkrati pretežno dobro odziven. Relativno enostavno je navidez utežiti kvadrokopter tako, da počasi zalebdi nad oznako in tam ostane, vendar potem ni dovolj odziven. V nasprotnem primeru, če kvadrokopter navidez razbremenimo, se lahko zelo hitro odziva. Vendar v tem primeru ponavadi pride do precejšnjega nihanja okoli oznake in v najslabšem primeru kvadrokopter oznako popolnoma izgubi. Najboljši parametri za krmilnik PID, ki smo jih uspeli najti za naš algoritem, so vidni v tabeli 4.1. Krmilniku PID smo dodali tudi dušilec. Ko izhod iz krmilnika pride na interval $[-\epsilon_d, \epsilon_d]$, se izhod še dodatno zmanjša. Na tak način krmilnik pošilja skoraj zanemarljivo majhne ukaze za premik, če se kvadrokopter nahaja približno nad oznako.

Proporcionalni dobiček Kp	0.6
Integralski dobiček Ki	0.08
Derivacijski dobiček Kd	0.2
Največja vrednost integrala i_{max}	0.2
Velikost dušilca ϵ_d	0.02

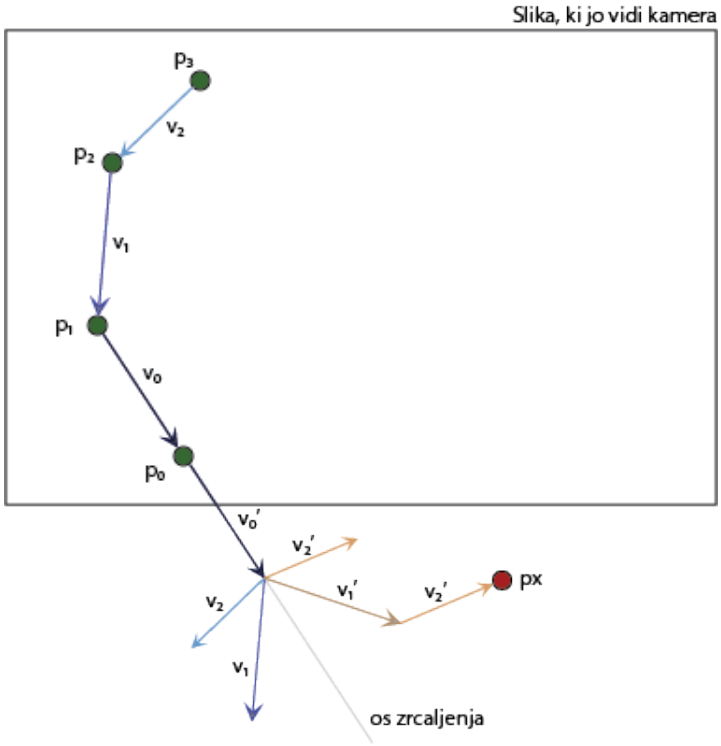
Tabela 4.1: Parametri za krmilnik PID.

4.3 Reševanje iz izgubljenega stanja

Krmiljenje je dokaj enostavno, če je oznaka v vidnem polju kamere. Kvadrokopter se lahko v tem primeru enostavno premakne proti njej. Vendar pa ne moremo zagotoviti, da bo oznaka vedno v vidnem polju. Zato smo implementirali tudi reševanje iz izgubljenega stanja. Algoritem v osnovi sloni na podobnih principih kot Kalmanov filter. Zapomne si n točk, kjer je nazadnje zaznal oznako in iz teh točk nato z vektorskimi operacijami ekstrapolira, kje oz. v kateri smeri naj bi se naslednja točka nahajala. To prikazujeta sliki 4.3 in 4.4.

Najprej se določi primarna smer, oz. os zrcaljenja. Ta os je določena z najbolj nedavno zaznanima točkama p_0 in p_1. Nato se poišče vektorje med vsemi točkami, kjer je bila zaznana oznaka. Ti vektorji so v_0, v_1 in v_2. Te vektorje se prezrcali čez os zrcaljenja in pomnoži z določeno konstanto oz. dobičkom, ki zmanjša težo vektorja. Tako dobimo v_0', v_1' in v_2'. Dobiček se zmanjšuje za vsak sledeči vektor, kar pomeni da bodo starejši položaji, kjer je bila zaznana oznaka, imeli manjšo težo na predikcijo novega položaja oz. na novo smer. Nazadnje se vse prezrcaljene ter zmanjšane vektorje sešteje skupaj in prišteje točki, kjer je bila nazadnje zaznana oznaka. Na tak način dobimo predikcijo p_x, kjer naj bi se izgubljena oznaka približno nahajala v naslednjem trenutku.

Zrcaljenje vektorjev je potrebno, ker predpostavljamo, da kvadrokopter dela krožne premike. V praksi se izkaže, da je to res, saj zaradi krmilnika PID ponavadi niha okoli oznake. Za dobiček, s katerim je pomanjšan vsak



Slika 4.3: Napoved izgubljene točke, ko je dobiček=1

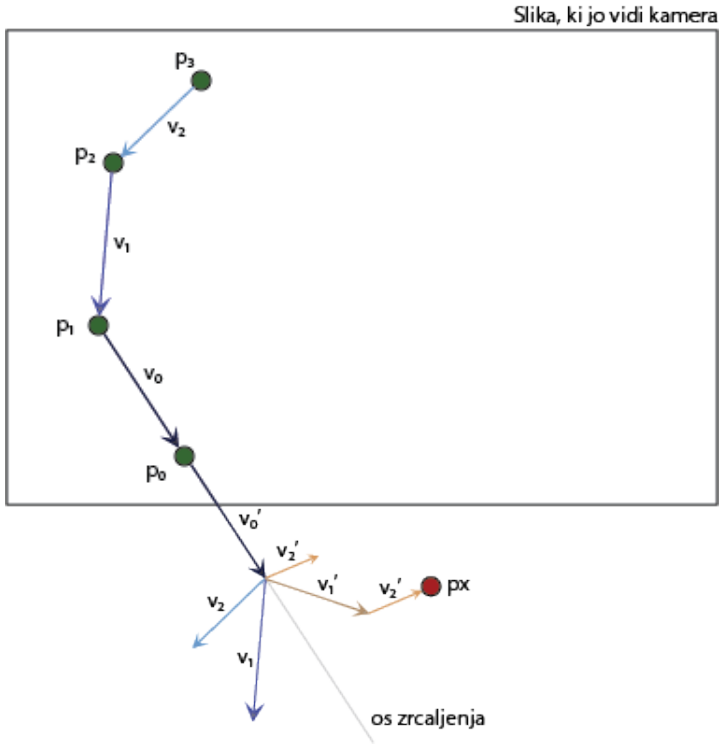
vektor, je predvidena vrednost med 0 in 1. V nalogi smo ga nastavili na 0.5. V splošnem le-ta skrbi za to, kako močno bo kvadrokopter zavil nazaj. Če dobiček nastavimo na 0, kvadrokopter ne bo zavil, ampak se bo premaknil v smer, kjer je nazadnje zaznal oznako.

Tako iskanje lahko predstavimo s formulo

$$p_x = p_0 + \sum_{i=0}^{n-1} k^i \cdot v_i', \tag{4.2}$$

kjer je p_x koordinata napovedane točke, p_0 koordinata zadnjega zaznanega položaja oznake, n število točk, ki smo si jih zapomnili za nazaj, k konstanta oz. dobiček in v_i' zrcaljen vektor med točkama p_i in p_{i+1}. Zrcaljenje vektorja lahko tudi predstavimo s formulo

$$v_i' = \frac{2(v_i \cdot v_0)}{\|v_0\|^2} v_0 - v_i, \tag{4.3}$$



Slika 4.4: Napoved izgubljene točke, ko je dobiček=0.75

kjer je v_i vektor med točkama p_i in p_{i+1} in v_0 primarni vektor.

Ker ne želimo, da se kvadrokopter zaleti v zid, oznako išče samo osem sekund. Če po preteku tega časa še vedno ne zazna oznake, se ustavi in čaka, da ga ročno potisnemo nad oznako.

Poglavje 5

Rezultati

Uspešnost razvitega sistema smo ovrednotili s tremi različnimi eksperimenti. Vsakega od teh eksperimentov smo izvedli petkrat, da smo dobili bolj verodostojne rezultate. V tem poglavju predstavimo dobljene rezultate.

5.1 Ocenjevanje višine

Najprej smo testirali kako natančno naš algoritem ocenjuje višino med kvadrokopterjem in oznako, v primerjavi z vgrajenim ultrazvočnim senzorjem. Kvadrokopter smo 30 sekund ročno držali približno 170 cm nad tlemi in beležili vrednosti, ki sta jih vračala ultrazvočni senzor in naš algoritem. Izvedli smo dve različici tega eksperimenta. Najprej smo oznako položili na tla, nato pa isti eksperiment izvedli še tako, da smo oznako nalepili na iRobot Roombo. Rezultate smo prikazali na slikah 5.1 in 5.2. Povprečne vrednosti vsakega poskusa lahko vidimo v tabelah 5.1 in 5.2. Poleg tega smo povprečne vrednosti prikazali tudi grafično, na slikah 5.3 in 5.4. Na slikah zelena vodoravna črta prikazuje povprečno izmerjeno višino v določenem poskusu, oranžna pa označuje pravilno vrednost oz. pravilno razdaljo med oznako in kvadrokopterjem.

Rezultati kažejo, da je naša metoda ocenjevanja višine bolj natančna od vgrajenega ultrazvočnega senzorja. Iz slik 5.1 in 5.2 lahko opazimo, da z vi-

	Ultrazvočni senzor [cm]	Standardni odklon (senzor) [cm]	Vizualno s kamero [cm]	Standardni odklon (vizualno) [cm]
Poskus 1	165.4	10.1	168.6	2.9
Poskus 2	137.3	2.2	164.0	2.0
Poskus 3	158.3	3.6	163.7	2.3
Poskus 4	76.1	6.1	167.3	2.4
Poskus 5	152.5	2.0	163.0	2.9
Povprečje	137.9	4.8	165.3	2.5

Tabela 5.1: Izmerjena višina med kvadrokopterjem in oznako, ko je oznaka na tleh. Dejanska višina kvadrokopterja nad oznako je bila tu 170 cm.

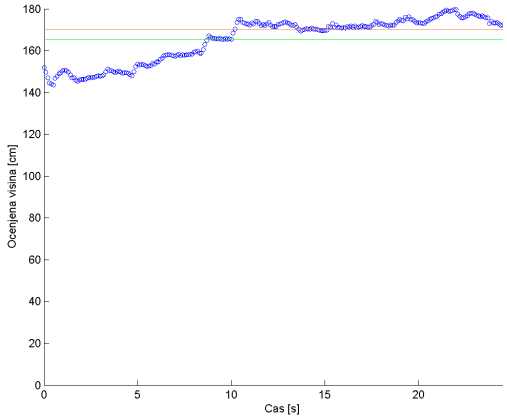
	Ultrazvočni senzor [cm]	Standardni odklon (senzor) [cm]	Vizualno s kamero [cm]	Standardni odklon (vizualno) [cm]
Poskus 1	95.9	4.3	132.3	13.5
Poskus 2	187.7	44.4	132.6	2.3
Poskus 3	75.0	9.8	129.6	2.5
Poskus 4	138.5	5.3	132.5	25.4
Poskus 5	135.0	14.2	132.3	7.0
Povprečje	126.4	15.6	131.9	10.1

Tabela 5.2: Izmerjena višina med kvadrokopterjem in oznako, ko je oznaka nameščena na iRobot Roombi. Dejanska višina kvadrokopterja nad oznako je bila tu 128 cm.

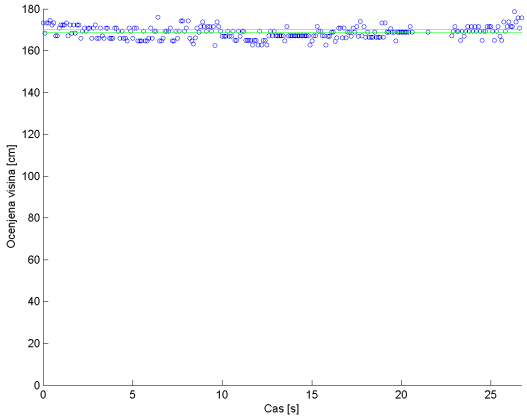
zualnim ocenjevanjem višine, dobimo tudi precej bolj konsistentne vrednosti, medtem ko ultrazvočni senzor lahko vrača različne vrednosti ob podobnih pogojih.

Poleg tega se izkaže, da za naše potrebe ultrazvočni senzor ni najbolj uporaben, saj meri razdaljo od kvadrokopterja do površine pod njim. Pravilne rezultate dobimo, če kvadrokopter lebdi točno nad Roombo. Če se premakne

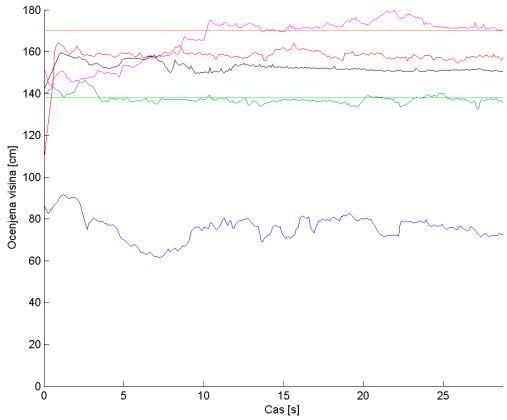
nekoliko vstran pa začne meriti razdaljo med kvadrokopterjem in tlemi in ne več med kvadrokopterjem in oznako. To lahko povzroči, da se kvadrokopter spusti in se ne dvigne več na pravo višino.



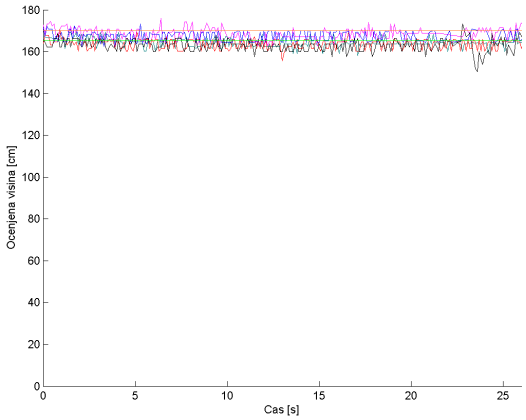
(a) Poskus 1 - meritve sonarja



(b) Poskus 1 - vizualne meritve

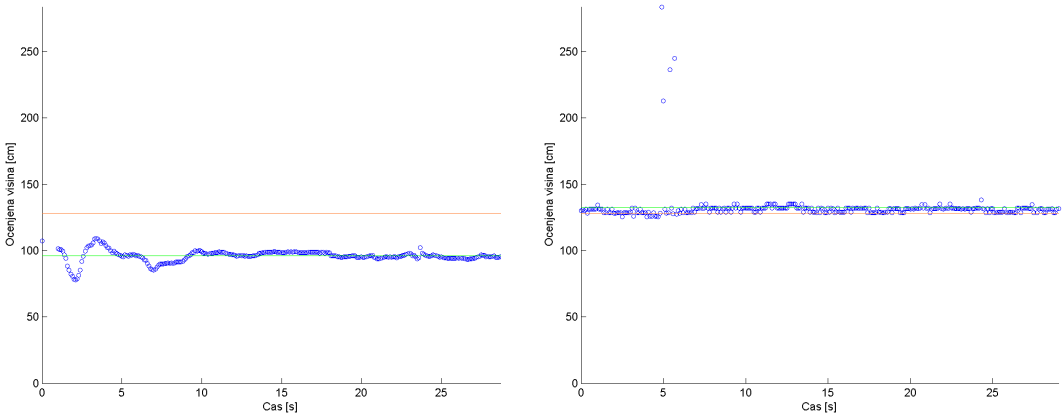


(c) Vsi poskusi skupaj - meritve sonarja



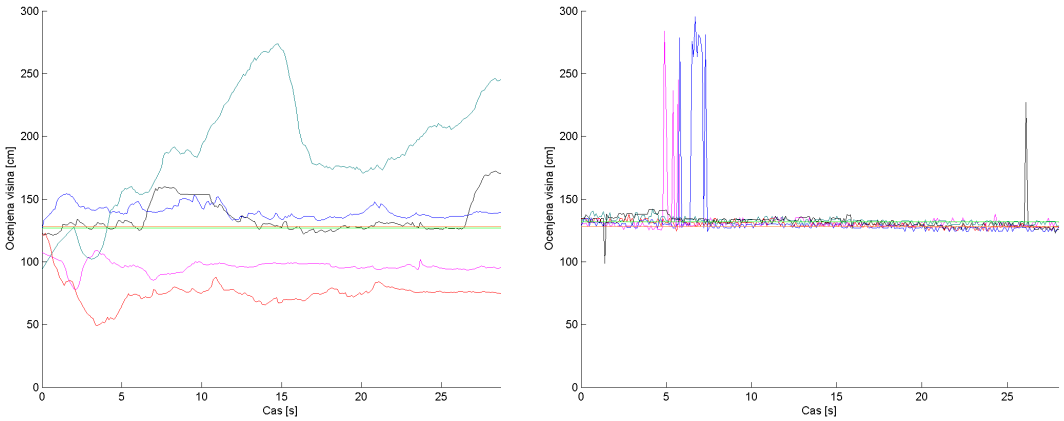
(d) Vsi poskusi skupaj - vizualne meritve

Slika 5.1: Izmerjena višina med kvadrokopterjem in oznako, ko je oznaka na tleh.



(a) Poskus 1 - meritve sonarja

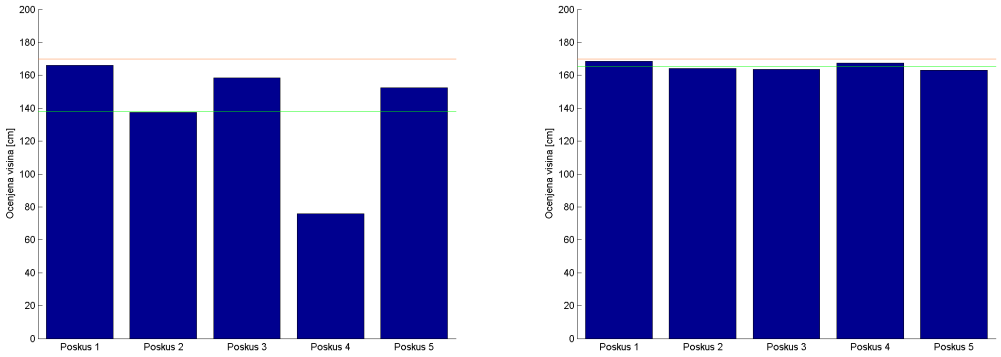
(b) Poskus 1 - vizualne meritve



(c) Vsi poskusi skupaj - meritve sonarja

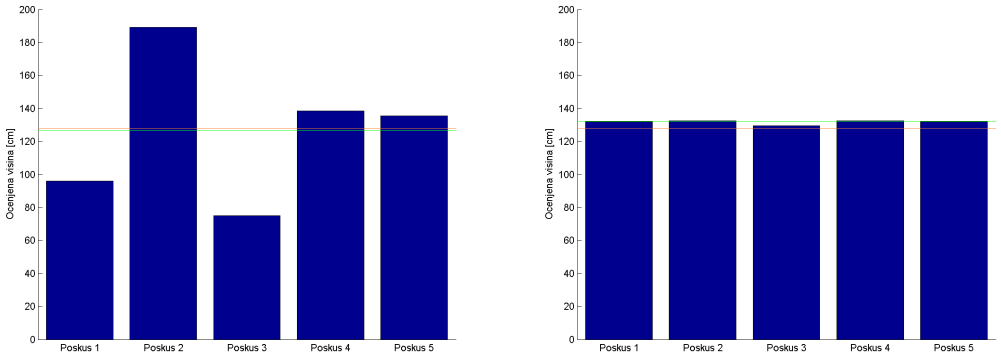
(d) Vsi poskusi skupaj - vizualne meritve

Slika 5.2: Izmerjena višina med kvadrokopterjem in oznako, ko je oznaka nameščena na iRobot Roombi.



(a) Povprečne višine - meritve sonarja (b) Povprečne višine - vizualne meritve

Slika 5.3: Povprečne višine posameznih poskusov, ko je oznaka na tleh.



(a) Povprečne višine - meritve sonarja (b) Povprečne višine - vizualne meritve

Slika 5.4: Povprečne višine posameznih poskusov, ko je oznaka nameščena na iRobot Roombi.

5.2 Stabilizacija nad stacionarno oznako

Z drugim eksperimentom, ki smo ga izvedli, smo ocenili kako natančno in robustno zna kvadrokopter lebdeti nad oznako, ki se ne premika, v primerjavi s privzeto kvadrokopterjevo stabilizacijo. Tu smo prav tako izvedli dve različici

eksperimenta. Najprej smo naš algoritem spremenili tako, da kvadrokopterju ni pošiljal ukazov za premike. Na tak način se je moral kvadrokopter zanašati na vgrajene metode za stabilizacijo. Pri tem uporablja spodnjo kamero, da določi teksturo tal in na podlagi podatkov o teksturi poskuša lebdeti na približno istem mestu.

V drugi različici smo testirali našo implementacijo sledenja objektom in stabilizacije. S pomočjo algoritmov opisanih v prejšnjih poglavjih, je kvadrokopter moral lebdeti čim bolj točno nad oznako. Povprečne rezultate poskusov smo prikazali v tabelah 5.3 in 5.4. Slike 5.5 prikazujejo oddaljenost kvadrokopterja od oznake, po oseh x in y (višina se ne upošteva). Zelena vodoravna črta označuje povprečno oddaljenost kvadrokopterja od oznake za določen poskus. Rdeče navpične črte označujejo, kdaj smo morali ročno posredovati in kvadrokopter potisniti nazaj nad oznako. V vsakem poskusu je kvadrokopter letel pet minut preden smo poskus prekinili.

Oceno za natančnost in robustnost AR.Dronovega privzetega ter našega algoritma za stabilizacijo lahko razberemo iz tabel 5.3 in 5.4. Natančnost lahko ocenimo na podlagi povprečne razdalje do oznake. Manjša kot je razdalja, bolj je algoritem natančen. Robustnost pa lahko ocenimo na podlagi števila ročnih posredovanj. Vsakič ko moramo posredovati pomeni, da se je kvadrokopter popolnoma izgubil in smo ga morali ročno prestaviti nazaj nad oznako. Večje kot je število ročnih posredovanj, manj je algoritem robusten. Poleg tega tabeli prikazujeta tudi razmerje med iteracijami, kjer je bila zaznana oznaka, proti iteracijam, kjer ni bila. To razmerje smo izračunali tako, da smo število iteracij, kjer je bila zaznana oznaka, delili s številom iteracij, kjer ni bila zaznana. Večje kot je to število, manjkrat je bila oznaka izgubljena in bolje deluje algoritem.

Iz rezultatov je razvidno, da je sistem, ki smo ga razvili, mnogo bolj robusten in natančen od privzete AR.Dronove stabilizacije. S privzeto stabilizacijo smo morali večkrat ročno posredovati, da je kvadrokopter ostal nad oznako. Z uporabo našega sistema pa to ni bilo potrebno. Občasno je še izgubil oznako iz vidnega polja, vendar jo je potem vedno znal ponovno najti. Natančnost

	Povprečna razdalja do oznake [cm]	Standardni odklon povprečne razdalje [cm]	Število ročnih posredovanj	Razmerje med iteracijami z zaznanimi in nezaznanimi oznakami
Poskus 1	35.8	19.8	21	1.82
Poskus 2	52.3	22.3	20	1.02
Poskus 3	45.8	20.5	28	0.88
Poskus 4	39.2	16.6	19	1.06
Poskus 5	50.3	22.6	21	1.55
Povprečje	44.7	20.3	21.8	1.27

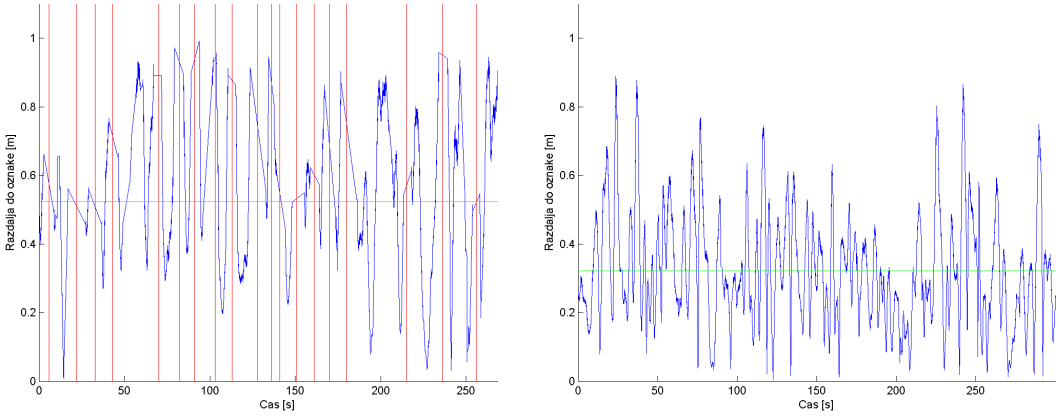
Tabela 5.3: Lebdenje nad stacionarno oznako, z AR.Dronovo privzeto stabilizacijo.

	Povprečna razdalja do oznake [cm]	Standardni odklon povprečne razdalje [cm]	Število ročnih posredovanj	Razmerje med iteracijami z zaznanimi in nezaznanimi oznakami
Poskus 1	23.7	13.5	0	32.59
Poskus 2	32.1	16.3	0	27.34
Poskus 3	30.9	16.5	0	13.70
Poskus 4	27.2	14.4	0	31.57
Poskus 5	34.1	18.5	0	14.27
Povprečje	29.6	15.8	0	23.89

Tabela 5.4: Lebdenje nad stacionarno oznako, s stabilizacijo implementirano v diplomski nalogi.

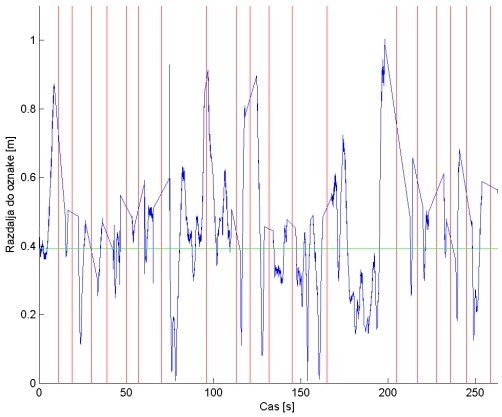
stabilizacije smo izboljšali za vsaj 33.8%. K natančnosti privzete stabilizacije prispeva predvsem dejstvo, da smo ročno posredovali, vsakič ko se je kvadrokopter premaknil stran od oznake. Brez posredovanja bi kvadrokopter enostavno odplaval stran.

Podali smo tudi grafa, prikazana na sliki 5.6. Sta enakih dimenzij kot slikovna ravnina kamere in prikazujeta, kje na sliki je bila zaznana oznaka[1]. Bolj kot so zaznave koncentrirane na eno mesto, bolj je stabilizacija natančna in manj kvadrokopter niha okoli oznake. Iz teh slik lahko opazimo, da z našim algoritmom kvadrokopter precej bolj natančno lebdi na mestu.

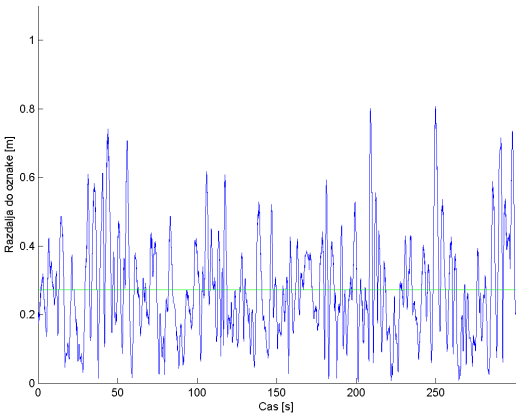


(a) Poskus 2 - privzeta stabilizacija

(b) Poskus 2 - izboljšana stabilizacija



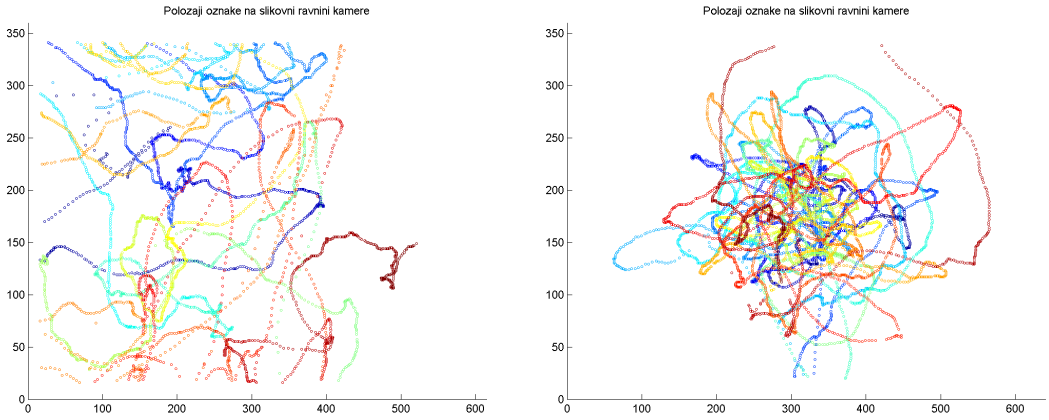
(c) Poskus 4 - privzeta stabilizacija



(d) Poskus 4 - izboljšana stabilizacija

Slika 5.5: Izmerjena razdalja od kvadrokopterja do oznake, ko je oznaka stacionarna.

[1]Prva detekcija je temno rdeča, zadnja je temno modra.



(a) Poskus 5 - privzeta stabilizacija (b) Poskus 1 - izboljšana stabilizacija

Slika 5.6: Zaznani položaj oznake na sliki, pridobljeni iz kvadrokopterjeve kamere. Oznaka se ne premika.

5.3 Sledenje premikajočemu se objektu

Zadnji eksperiment, ki smo ga izvedli, je meril natančnost in robustnost sledenja premikajočemu se objektu. Podobno kot pri drugem eksperimentu je moral kvadrokopter lebdeti čim bolj točno nad oznako, vendar se je v tem eksperimentu oznaka premikala. Oznako smo nalepili na iRobot Roombo in jo krmilili tako, da je njena pot orisala kvadrat s stranico dolgo približno en meter. Hitrost Roombe smo nastavili na 0.2 m/s. Izračunali smo iste statistike in podali iste grafe kot za drugi eksperiment. Prikazane so v tabeli 5.5 in slikah 5.7 ter 5.8.

Rezultati za premikajoči se objekt so boljši od privzete stabilizacije, vendar so nekoliko slabši od rezultatov za stacionarni objekt. Natančnost se je zmanjšala za 26.8%. Prav tako se je nekoliko zmanjšala tudi robustnost. Do zmanjšanja natančnosti pride zaradi krmilnika PID. Parametri, ki smo jih dobili z umerjanjem krmilnika, bolje stabilizirajo kvadrokopter, vendar se nekoliko počasneje odzivajo na spremembe. To pomeni da vsakič, ko se Roomba

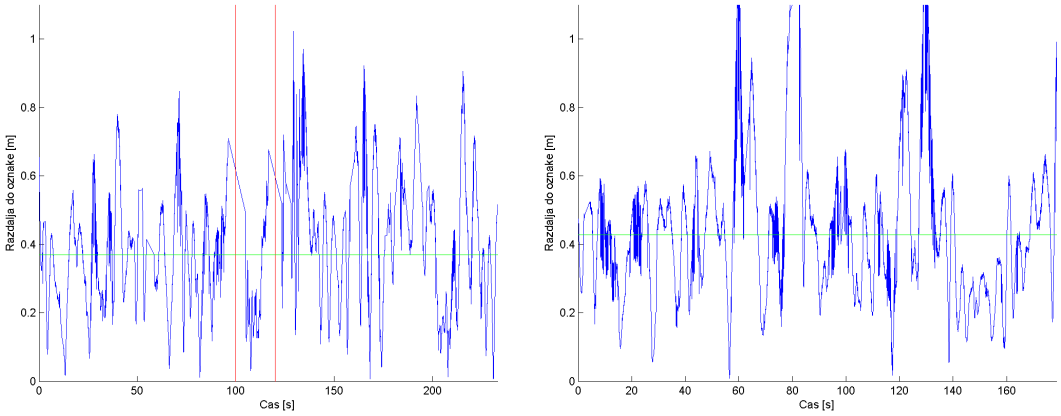
	Povprečna razdalja do oznake [cm]	Standardni odklon povprečne razdalje [cm]	Število ročnih posredovanj	Razmerje med iteracijami z zaznanimi in nezaznanimi oznakami
Poskus 1	36.9	17.6	2	3.66
Poskus 2	42.7	19.7	0	10.00
Poskus 3	40.0	22.5	0	6.31
Poskus 4	42.3	21.4	3	3.42
Poskus 5	39.8	18.8	0	5.81
Povprečje	40.4	20.0	1	5.84

Tabela 5.5: Sledenje premikajočemu se objektu, z uporabo metod opisanih v diplomskem nalogi.

ustavi ali požene, kvadrokopter nekoliko bolj zaniha okoli oznake, kot če bi Roomba stala pri miru ali se premikala s konstantno hitrostjo. Robustnost se zmanjša, ker algoritem za reševanje iz izgubljenega stanja predpostavlja, da oznaka ne spreminja smeri, oz. spreminja smer s konstantnim odklonom. Občasno se zgodi, da kvadrokopter izgubi oznako iz vidnega polja in preden jo spet najde, Roomba zavije stran, kar povzroči, da se kvadrokopter popolnoma izgubi.

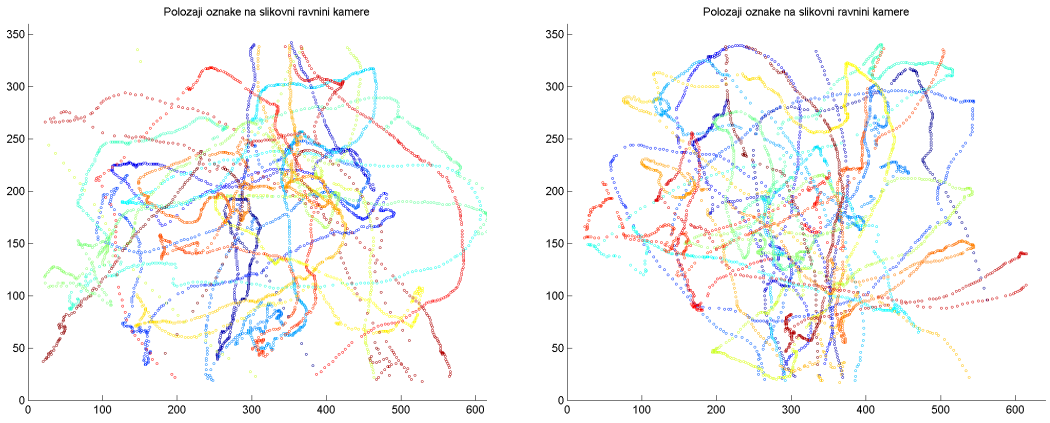
Poleg tega smo vizualizirali tudi položaj kvadrokopterja glede na iRobot Roombo. To lahko opazimo na slikah 5.9. Modra črta prikazuje položaj Roombe. Barvne točke prikazujejo položaj kvadrokopterja[2]. Zaradi preglednosti smo vizualizirali samo del določenih poskusov. Sicer je v vsakem poskusu Roomba naredila pet obhodov, preden smo poskus ustavili.

[2]Pot kvadrokopterja poteka od temno rdeče proti temno modri. Roomba je krožila v nasprotni smeri urinega kazalca.



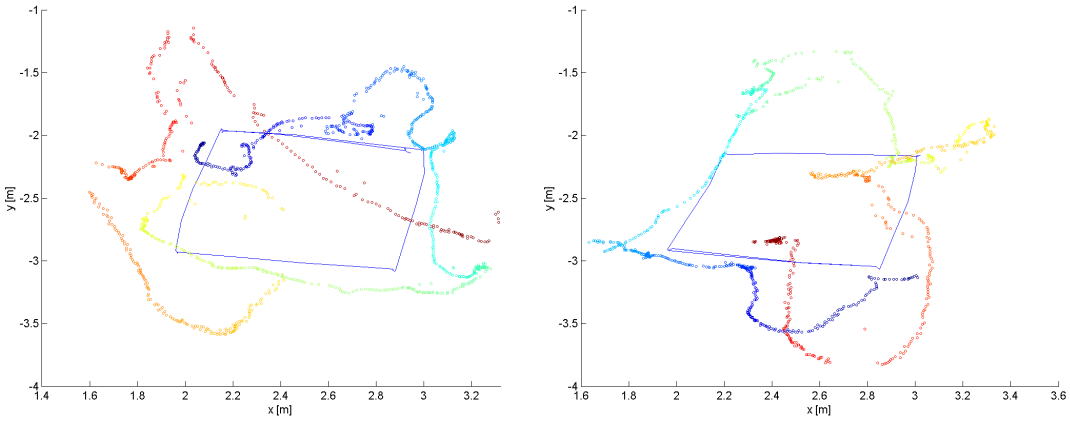
(a) Poskus 1 (b) Poskus 2

Slika 5.7: Izmerjena razdalja od kvadrokopterja do oznake, ko se oznaka premika.



(a) Poskus 4 (b) Poskus 5

Slika 5.8: Zaznani položaj oznake na sliki, pridobljeni iz kvadrokopterjeve kamere. Oznaka se premika.



(a) Del poskusa 4 (b) Del poskusa 5

Slika 5.9: Položaj kvadrokopterja in iRobot Roombe v prostoru.

Poglavje 6

Sklep

V diplomski nalogi smo implementirali sistem, s katerim lahko kvadrokopter Parrot AR.Drone sledi označenemu objektu. Program teče v okolju ROS in temelji na paketu ardrone_autonomy.

Zaznavanje oznake je izvedeno tako, da se sliko iz kamere najprej binarizira in na njej določi vse like. Med temi liki se določi kateri so kvadratne oblike. Na podlagi tega se nato ugotovi, kje na sliki se nahaja oznaka. Nazadnje se položaj oznake preslika iz slikovnih koordinat v globalne in pošlje naprej vozlišču, ki skrbi za krmiljenje kvadrokopterja.

Premikanje je izvedeno s krmilnikoma PID. Ker kvadrokopter občasno izgubi oznako iz vidnega polja, še posebej če se oznaka premika, smo implementirali algoritem za reševanje iz izgubljenega stanja. Deluje tako, da si zapomni zadnjih deset točk, kjer je videl oznako in na podlagi tega ekstrapolira, kje bi se morala oznaka pojaviti v naslednji iteraciji.

Da bi minimizirali izgubljanje oznake, smo poskušali zagotoviti, da se kvadrokopter vedno drži na minimalni višini 1.5 m nad oznako. Poleg tega smo tudi modificirali sprednjo kamero tako, da kaže navzdol in uporabili le-to namesto spodnje kamere, saj ima sprednja kamera širši zorni kot.

Rezultati kažejo, da kvadrokopter bolj stabilno lebdi na mestu, če uporablja algoritem implementiran v tej nalogi namesto svoje privzete stabilizacije.

Prav tako lahko relativno robustno sledi premikajočemu se objektu, če je le-ta označen s preprosto oznako. Ugotovili smo tudi, da je ocenjevanje višine, na podlagi velikosti oznake na sliki, bolj natančno od višine, izmerjene s pomočjo ultrazvočnega senzorja, nameščenega na kvadrokopterju.

6.1 Možne izboljšave

Največji problem pri zaznavanju oznake je, da višje kot se kvadrokopter povzpne, težje zazna oznako. Vendar je zaželjeno, da lebdi dovolj visoko, saj je tako bolj stabilen. Do tega pride, ker ne more več razločevati med zunanjim in notranjim robom oznake in ju zazna skupaj kot eno črto. Možno bi bilo preklopiti sprednjo kamero na višjo ločljivost. Privzeto ardrone_autonomy nastavi ločljivost za obe kameri na 640x360 slikovnih elementov. Morda bi bilo možno tudi drugače binarizirati sliko ali pa uporabiti drugačno oznako, da bi bila robova bolje ločena.

Nekoliko problematično je tudi reševanje iz izgubljenega stanja. Algoritem, ki smo ga predstavili v nalogi, sicer v večini primerov deluje, občasno se pa še vedno zgodi, da izgubi oznako. Izboljšali bi ga lahko tako, da bi poleg sledenja hkrati tudi izvajali simultano lokalizacijo in mapiranje ali SLAM (ang. Simultaneous Localization And Mapping) [13][16]. Trenutno kvadrokopter ne ve, kje v prostoru je zaznal oznako. Ve samo, kje je zaznal oznako relativno nase. Problem z mapiranjem je, da navzdol obrnjena kamera ne zazna veliko značilk, s katerimi bi se lahko kvadrokopter orientiral.

Poleg tega bi se zagotovo dalo izboljšati tudi stabilizacijo, da bi kvadrokopter manj nihal okoli oznake. Za to bi bilo potrebno nadaljno umerjanje parametrov krmilnika PID.

Boljše rezultate bi lahko dosegli tudi z z uporabo bolj kvalitetnega kvadrokopterja, z natančnejšimi senzorji in bolj odzivnim ter natančnim krmilnim mehanizmom.

Literatura

[1] Karl J. Åström and Tore Hägglund. PID controllers: theory, design, and tuning. *Instrument Society of America, Research Triangle Park, NC*, 1995.

[2] AutonomyLab. ardrone_autonomy, 2014. `https://github.com/AutonomyLab/ardrone_autonomy` [dostop: 15.8.2014].

[3] Y. Li, K.H. Ang, and G.C.Y. Chong. PID control system analysis and design. *IEEE Control Systems Magazine*, 26(1):32–41, February 2006.

[4] Lucidity. Cool drones don't look at explosions, 2011. `http://roswellflighttestcrew.typepad.com/blog/2011/08/cool-drones-dont-look-at-explosions.html` [dostop: 30.8.2014].

[5] NOAA. Unmanned drones to probe hurricanes, 2006. `http://uas.noaa.gov/news/msnbc-unmanned-drones.html` [dostop: 30.8.2014].

[6] Parrot. Technical specifications, 2014. `http://ardrone2.parrot.com/ardrone-2/specifications/` [dostop: 15.8.2014].

[7] Urs Ramer. An iterative procedure for the polygonal approximation of plane curves. *Computer Graphics and Image Processing*, 1(3):244 – 256, 1972.

[8] Andreas Raptopoulos. No roads? There's a drone for that, 2013. http://www.ted.com/talks/andreas_raptopoulos_no_roads_there_s_a_drone_for_that [dostop: 30.8.2014] TED (conference).

[9] ROS. Core Components, 2014. http://www.ros.org/core-components/ [dostop: 15.8.2014].

[10] David Sellers. An overview of proportional plus integral plus derivative control and suggestions for its successful application and implementation. In *Proceedings for the 2001 International Conference on Enhanced Building Operations*, 2001.

[11] Jurij Slabanja. tracking_ardrone, 2014. https://bitbucket.org/meerkqat/tracking_ardrone/src [dostop: 15.8.2014].

[12] Satoshi Suzuki and Keiichi Abe. Topological structural analysis of digitized binary images by border following. *Computer Vision, Graphics, and Image Processing*, 30(1):32–46, 1985.

[13] Sebastian Thrun and JohnJ. Leonard. Simultaneous Localization and Mapping. In Bruno Siciliano and Oussama Khatib, editors, *Springer Handbook of Robotics*, pages 871–889. Springer Berlin Heidelberg, 2008.

[14] K.P. Valavanis. Coordination of Helicopter UAVs for Aerial Forest-Fire Surveillance. In *Applications of Intelligent Control to Engineering Systems*, Intelligent Systems, Control and Automation: Science and Engineering. Springer, 2009.

[15] vp153. calibration.cpp, 2014. https://code.ros.org/trac/opencv/browser/trunk/opencv/samples/cpp/calibration.cpp?rev=3474 [dostop: 15.8.2014].

[16] Chieh-Chih Wang and C. Thorpe. Simultaneous localization and mapping with detection and tracking of moving objects. In *Robotics and Automa-*

tion, 2002. Proceedings. ICRA '02. IEEE International Conference on, volume 3, pages 2918–2924, 2002.

[17] Wikipedia. Aerosonde Laima bottom view, 2014. `http://en.wikipedia.org/wiki/File:Aerosonde_Laima_bottom_view.jpg` [dostop: 30.8.2014].

[18] Wikipedia. Parrot AR.Drone, 2014. `http://en.wikipedia.org/wiki/Parrot_AR.Drone` [dostop: 15.8.2014].